UNIVERSITÀ DEGLI STUDI DI BRESCIA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Applicazione di un laser range scanner a un robot mobile

Studente:

Giovanni Mattei Matricola: **70419** **Firma**

Ente ospitante:

Laboratorio di robotica avanzata

Relatore:

Chiar.mo Prof. Riccardo Cassinis **Firma**

ANNO ACCADEMICO 2009/2010

Sommario

Argomento trattato

Il lavoro verte sull'uso e sull'integrazione di un pacchetto software utilizzabile nell'ambito della robotica mobile. Verrà descritta l'installazione di un calcolatore e di un laser su un robot, affinché possa essere in grado di muoversi e localizzarsi all'interno di ambienti dei quali sarà possibile anche, mediante il laser, tracciare mappe (capitoli 1 e 2). Verranno inoltre presentati il processo di costruzione di una stazione di ricarica per il robot (capitolo 3) e la creazione di una pagina web per l'interazione a distanza (capitolo 4).

Introduzione

Il lavoro svolto nasce dall'interesse per l'applicazione dell'informatica alla robotica mobile. L'ente ospitante, ossia il laboratorio di robotica avanzata della Facoltà di Ingegneria di Brescia, ha a disposizione robot in grado di effettuare videosorveglianza, una della applicazioni più interessanti della robotica mobile. Dopo averne visto il funzionamento, si è pensato di potenziare le funzioni di un altro robot appartenente sempre all'ente ospitante.

Il lavoro svolto si è basato soprattutto sull'uso di due pacchetti software: ARIA e ARNL. Essi sono necessari per poter interagire con diversi tipi di robot. Il primo contiene il codice necessario a far compiere al robot azioni basilari, come i movimenti; il secondo permette invece al robot di compiere azioni più sofisticate, come localizzarsi all'interno di un ambiente o tracciare mappe mediante un laser.

La parte successiva del lavoro è consistita nella costruzione di una stazione di ricarica indispensabile per poter comandare il robot a distanza, vista la necessità di mettere in condizione il robot di ricaricare autonomamente le batterie.

La parte conclusiva del lavoro si è sviluppata nell'ambito della programmazione shell per la creazione di CGI che, invocate da un'apposita interfaccia web, possano dare automaticamente i comandi necessari al robot per muoversi.

Capitolo 1

Installazione del calcolatore e del laser sul robot

Situazione di partenza

La situazione iniziale di partenza è data da un robot (Figura 1) con le seguenti caratteristiche:

- Marca: MobileRobots
- Modello: Pioner 1
- Numero di ruote motrici: 2
- Numero di ruote non motrici: 1
- Sistema di rilevamento ostacoli: sonar anteriori e laterali
- Batteria: 12 volt
- Metodo di comunicazione: via radio con un calcolatore fisso
- Nome: Tobor

Figura 1: Tobor

Obbiettivi da raggiungere nella prima parte

Gli obbiettivi che ci si propone di raggiungere con questo progetto consistono nell'apportare modifiche alla struttura del robot e al metodo di comunicazione.

Alla fine del lavoro il robot dovrà essere in grado di rilevare gli ostacoli non solo mediante sonar ma anche grazie all'ausilio di un laser che gli permetta di avere a disposizione rilevazioni molto più precise e corrette. Il robot verrà inoltre direttamente controllato da un calcolatore di dimensioni ridotte (un netbook) montato direttamente e fisicamente su di esso. Tutto ciò da una parte consentirà al robot di avere una mobilità molto più ampia e, dall'altra, di veder ridotto il rischio di non avvertire la presenza di ostacoli.

Caratteristiche del calcolatore da installare sul robot

- Marca: Acer
- Modello: Aspire One
- Processore: Intel Atom N270
- Ram: 512 MB DDR2
- Storage: 8 GB SSD
- OS: Linux

Modifiche di tipo software da effettuare sul calcolatore

Al fine di consentire al calcolatore di poter inviare comandi al robot, è stato necessario apportare alcune modifiche di tipo software. Il robot dispone di una porta seriale come connettore. Fino ad ora essa era collegata a un trasmettitore radio che riceveva i comandi da un calcolatore fisso. Siccome il calcolatore da installare sul robot, essendo recente, non possedeva una porta seriale, si è reso necessario installare un adattatore che simulasse una porta seriale pur utilizzando

fisicamente una porta usb. Si è poi provveduto ad installare le librerie necessarie al funzionamento del robot, disponibili sul sito della “Mobile Robots” e raccolte nella suite denominata “ARIA”. Essa offre numerosi programmi oltre alle sopra citate librerie. Fra di essi, due in particolare sono stati utilizzati: MobileSim (Figura 2) e Mapper3basic. Il primo è un simulatore. Il suo compito è quello di simulare il comportamento di un robot ActivMedia, come appunto Tobor, all’interno di una mappa già pronta o creata dall’utente stesso con il secondo programma.

MobileSim permette di iniziare a programmare utilizzando “ARIA” senza dover impiegare un vero e proprio robot. Infatti i programmi costruiti basandosi su ARIA, effettuano il primo tentativo di connessione col simulatore e non col robot fisico. MobileSim è utile inoltre per osservare e quindi correggere eventuali comportamenti sbagliati del robot. Alcune caratteristiche che sono disponibili sul simulatore, però, non sono presenti nella realtà. Ad esempio, il robot virtuale utilizzato da MobileSim possiede sonar sia anteriori, sia posteriori. Tobor, invece, possiede solo quelli anteriori. I comportamenti del robot virtuale saranno quindi, per certi aspetti, diversi da quello reale.

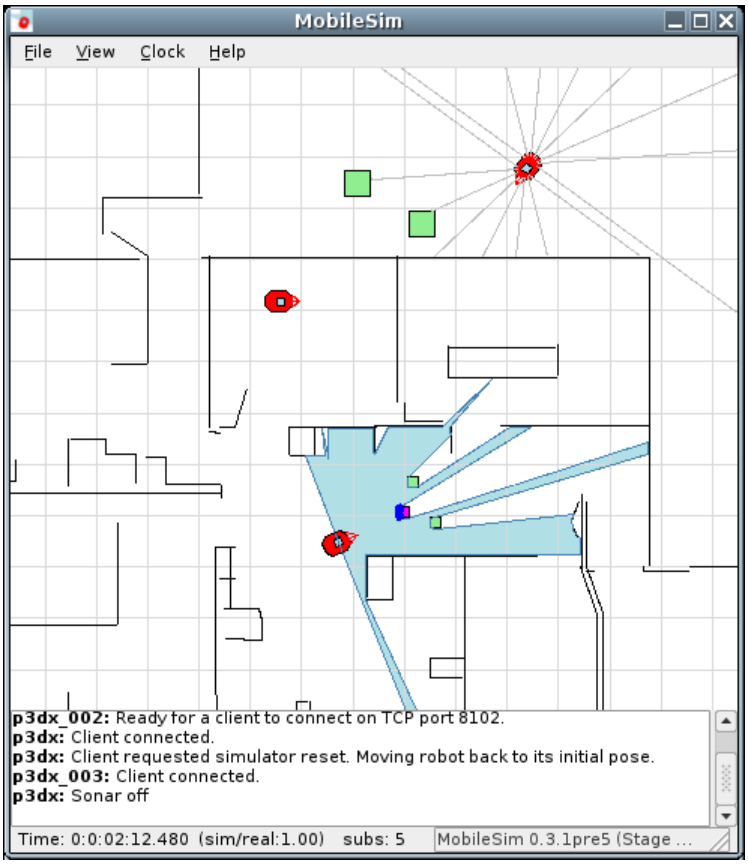

Figura 2: il programma MobileSim

In figura 2 sono presenti tre robot (di colore rosso) che si muovono all'interno di una mappa. Quello in alto si muove con l'ausilio di sonar (rappresentati dai raggi che si dipartono dal robot stesso). Quello in mezzo è fermo, mentre quello in basso possiede un laser la cui azione è rappresentata dall' "ombra" azzurra.

Mapper3basic è invece un software per disegnare mappe di ambienti. Ha un'interfaccia molto semplice che permette di tracciare linee su sfondo bianco. Potrebbe essere utile nel caso in cui non si disponesse di software avanzati per la creazione di mappe mediante laser o simili, oppure per fare delle prove di creazione di mappe da passare poi a MobileSim.

E' stato infine necessario installare sul netbook di Tobor il servizio server SSH (acronimo di secure shell). Grazie a quest'ultimo è infatti possibile lavorare da qualunque altro calcolatore in ogni parte del mondo con la stessa efficienza che si avrebbe se si interagisse direttamente con la macchina (a condizione che sia assicurata la velocità di connessione). In questo modo è possibile far eseguire i programmi da remoto, invece che dal calcolatore di Tobor. In Figura 3 vi è una rappresentazione schematica dei vari componenti che entrano in gioco.

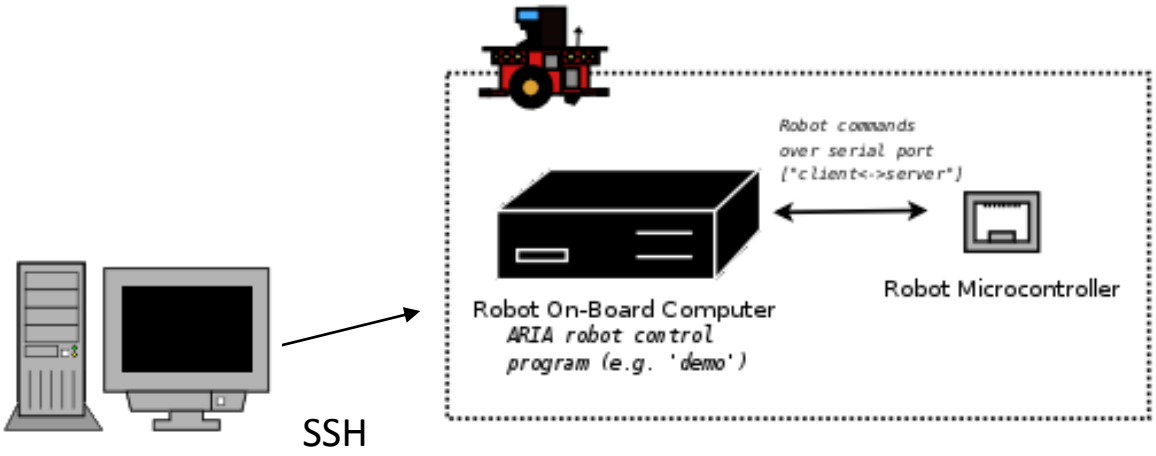

Figura 3: Un qualunque PC si puo' connettere via SSH al calcolatore di Tobor, il quale a sua volta controlla il microcontrollore del robot.

Le classi fondamentali di Aria

Aria è un pacchetto software scritto in C++ nato come interfaccia per i robot ActiveMedia (MobileRobots). Con Aria è possibile interagire col robot in due modi: inviando semplicemente comandi al robot mediante la classe ArRobot oppure facendo uso delle Action, metodo più sofisticato. Mediante la classe ArNetworking, che sfrutta i protocolli TCP e UDP, è inoltre possibile interagire col robot attraverso la rete. Le classi e i metodi di Aria si fanno dunque carico di compiere le azioni di controllo necessarie a una corretta navigazione del robot: individuare ed evitare gli ostacoli, interagire con i vari sensori (laser, sonar, paraurti) e localizzarsi all'interno di mappe di ambienti (questo mediante la libreria ausiliaria chiamata Arnl).

Il cuore di Aria è costituito quindi dalla classe ArRobot. Essa si occupa di interagire col firmware del robot: riceve tutte le informazioni dai sensori, le elabora e, interagendo con eventuali Action, di cui si parlerà dopo, invia i comandi di movimento al robot. Questo permette una navigazione "intelligente" del robot.

Quando si scrive un programma utilizzando Aria, quindi, la prima cosa da fare è stabilire una connessione col robot attraverso un oggetto della classe ArRobotConnector. Esso si occupa anche di elaborare gli eventuali parametri passati dalla linea di comando al momento dell'invocazione del programma. Analogamente, per connettersi ad un laser, si utilizza un oggetto della classe ArLaserConnector. Un oggetto della classe ArRobotConnector fa il suo primo tentativo di connessione con un simulatore (es. MobileSim). Nel caso quest'ultimo non fosse in esecuzione, allora viene stabilita una connessione col robot.

Ad un robot possono essere collegati uno o più accessori. L'oggetto ArRobotConnector necessita quindi di informazioni su quale device è connesso al robot (o al calcolatore del robot). Esse possono essere prese sia dal file di parametri relativo al modello del robot utilizzato, oppure dalla riga di comando, come già anticipato. Nel primo caso, il file di parametri viene letto subito dopo aver stabilito la connessione col robot. Nel secondo caso interviene in ausilio un oggetto della classe ArArgumentParser che si occupa appunto di memorizzare tutti i parametri passati dalla riga di comando. Ad esempio, per indicare la porta a cui è connesso un eventuale laser bisogna digitare "-lp" (che sta per laser port) seguito da una porta.

Il sistema di comunicazione fra il calcolatore, su cui è in esecuzione il programma che utilizza Aria, e il robot, è basato sullo scambio di pacchetti. Ogni comando inviato al robot corrisponde all'invio

di uno o più pacchetti. In risposta, il robot invia ogni 100 millisecondi informazioni (SIP, ossia server information packets) sul suo stato (velocità, carica della batteria, letture dei sonar), informazioni che possono essere sfruttate invocando i diversi metodi "get" della classe ArRobot. Infatti, quello che accade ogni 100 millisecondi è in realtà un ciclo chiamato ArRobot Task Cycle (Figura 4).

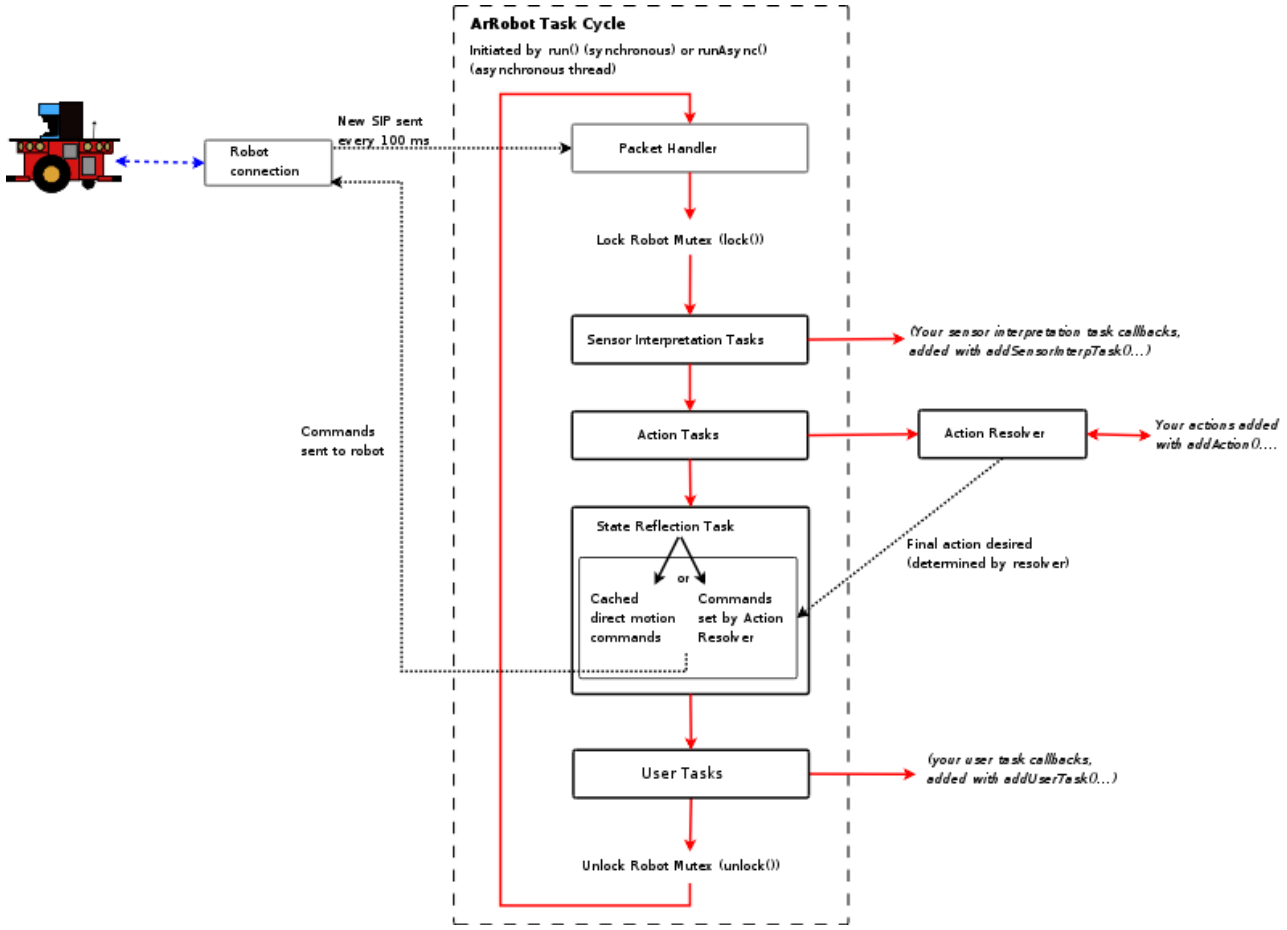

Figura 4: L' ArRobot Task Cycle

Questo ciclo puo' essere avviato in due modi: sincrono (attraverso il metodo "ArRobot::run()") o asincrono, cioè creando un thread (una parte di processo) indipendente (attraverso il metodo "ArRobot::runAsync()"). Vi possono quindi essere uno o più thread, e quindi uno o piu cicli, che interagiscono contemporaneamente col robot. Risulta pertanto necessario, all'inizio di ogni ciclo, attraverso il metodo "ArRobot::lock()", bloccare le variabili da modificare affinchè nessun altro

ciclo possa andarle a modificare nel frattempo. Se non si facesse così, si avrebbero problemi di "race condition", ossia di accessi simultanei di due o più thread alla stessa variabile.

Come già accennato prima, la classe ArRobot tiene memorizzate istantaneamente tutte le informazioni sul robot come ad esempio la posizione, la velocità delle ruote, il voltaggio della batteria, le rilevazioni dei sonar. Ci sono molti metodi get per ottenere ed eventualmente sfruttare queste informazioni. Per conoscere la velocità, ad esempio, c'è il metodo "ArRobot::getVel()". Queste informazioni possono essere modificate servendosi di due modalità: i comandi diretti inviati o le Action.

I comandi diretti consistono in pacchetti da 1 byte più eventuali argomenti. Tali pacchetti vengono passati al firmware del robot. Per esempio, il comando numero 4 (ENABLE) con l'argomento 1 attiva i motori del robot, mentre con l'argomento 0 li disattiva. Vi sono poi i comandi diretti di movimento. Ad esempio "ArRobot::setVel()" serve per regolare la velocità traslazionale, mentre "ArRobot::setRotVel()" quella rotazionale. "ArRobot::setHeading()" serve per regolare l'angolo di rotazione in caso di svolta a destra o a sinistra, "ArRobot::move()"per far percorrere al robot una data distanza (passata come argomento) e "ArRobot::stop()" per fermare i movimenti del robot. Tutti questi comandi, come già detto, vanno a modificare tutte quelle informazioni di cui tiene memoria la classe ArRobot. Essa svolge la funzione di trasferirle al firmware del robot alla fine di ogni ciclo.

Le Action invece sono utili quando si vuole che il robot abbia comportamenti avanzati, non ottenibili con l'utilizzo dei comandi diretti che, essendo semplici, non permettono di ottenere nulla di troppo sofisticato. Le Action sono oggetti individuali che si occupano ognuno di regolare i movimenti del robot in una determinata situazione. Una o più Action combinate, permettono quindi al robot di avere comportamenti avanzati. Ogni Action è costruita come sottoclasse della classe "ArAction" (da cui eredita il metodo "fire()", responsabile dell'esecuzione dell'azione) e puo' essere collegata a un oggetto "ArRobot" mediante il metodo "ArRobot::addAction()", passandogli come argomento un numero intero che rappresenta la priorità dell'azione. Più questo numero è grande, più l'azione acquista importanza. Se quindi vi sono più azioni agenti nello stesso programma, viene valutata per prima quella col numero di priorità maggiore. Questo compito è affidato all' "Action resolver" ad ogni ciclo. Esso valuta insieme tutte le Action e ne produce una sola che sarà poi passata come argomento al metodo "ArAction::fire()". Ciò che ne risulta è dunque un solo comando di movimento al robot. Bisogna prestare particolare attenzione

nell'utilizzo di eventuali comandi diretti di movimento (come ad esempio "ArRobot::setVel()") quando sono attive anche delle Action: potrebbero infatti verificarsi conflitti e comportamenti indesiderati nel robot. Le Action sono utili soprattutto nel caso si debba far navigare il robot all'interno di una stanza facendogli evitare gli ostacoli. Se ad esempio il laser o il sonar o entrambi rilevano che c'è un ostacolo a una data distanza, interviene una Action che fa in modo che venga evitato. E' sufficiente infatti agire sulla velocità delle ruote perché, anche nel caso di dover far compiere al robot una rotazione a destra, ciò che viene fatto è in realtà bloccare la ruota di destra e far girare solo quella di sinistra (in base all'angolo di rotazione scelto, anch'esso impostabile). Il programma "Caramelle", di cui si parlerà tra poco, sfrutta proprio il principio di più Action che agiscono contemporaneamente. E' possibile anche creare gruppi di azioni istanziando la classe "ArActionGroup". Questo permette di abilitare o disalbilitare più Action contemporaneamente.

Le Action servono proprio per reagire a situazioni che potrebbero portare il robot a bloccarsi. Questo è reso possibile grazie anche alla presenza dei "Range devices". Essi sono astrazioni dei sensori che permettono di individuare ostacoli grazie a continue rilevazioni dell'ambiente ciscostante. La classe che rappresenta tutti i Range devices è "ArRangeDevice". Vi sono poi le classi figlie, una per ogni strumento annesso al robot. C'è quella per i sonar (ArSonarDevice), quella per il laser (ArLaser e le sue sottoclassi come ArUrg), quella per i paraurti (ArBumpers) e quella per i sensori a raggi infrarossi (ArIRs). Vi è inoltre un Range device virtuale che ha il compito di individuare le "forbidden areas", ossia le zone proibite della mappa, zone aggiunte manualmente a cui il robot non deve accedere. La classe che lo rappresenta è "ArForbidenRangeDevice", quella per le mappe è "ArMap". Le letture effettuate dai sensori vengono raccolte da oggetti "ArRangeBuffer" che possono essere di due tipi: correnti, ovvero che contengono le rilevazioni più recenti, o cumulativi, cioè che contengono rilevazioni per un periodo più lungo. Per sfruttare le rilevazioni dei sensori, è necessario dichiararli esplicitamente nel programma e collegarli poi all'oggetto ArRobot, da cui possono ottenere tutte le informazioni ad essi necessarie. Per i sonar, i paraurti e i sensori a raggi infrarossi è sufficiente invocare il metodo "ArRobot::addRangeDevice()" passandogli come argomento il range device. Per il laser invece è necessario utilizzare una classe apposita (ArLaserConnector), che funziona in modo analogo alla classe ArRobotConnector. Questo perché il laser non è integrato nel microcontrollore del robot come i sonar.

Aria è scritto in modo tale da accettare anche eventuali input da tastiera. La classe che si occupa di gestirli è “ArKeyHandler”. Esiste anche una Action che permette di far navigare il robot all’interno della stanza utilizzando i pulsanti delle frecce. Essa si chiama ArActionKeydrive.

E’ doveroso citare anche le classi necessarie a realizzare il multi-threading. La classe che si occupa di garantire la mutua esclusione è “ArMutex”. E’ proprio questa che interviene quando viene invocato il metodo “lock” nell’ “ArRobot Task Cycle” impedendo che più thread scrivano e accedano alla stessa variabile: solo un thread alla volta puo’ farlo. Se non ci fosse questo meccanismo, verrebbero letti dati non corretti oppure sarebbero sovrascritti dati non ancora letti e di cui c’è bisogno. Si potrebbero quindi verificare, sempre nel caso di più thread che operino contemporaneamente, seri conflitti di accesso alla stessa variabile. E’ chiaro che quando un thread ha finito di scrivere su una variabile, deve renderla di nuovo disponibile, altrimenti verrebbero a crearsi situazioni di deadlock.

La classe “ArCondition” si occupa invece di interrompere,se necessario, il lavoro di un thread, ed eventualmente di farlo riprendere dopo un certo periodo di tempo.

Primi problemi incontrati

I primi tentativi sono stati effettuati con un laser di tipo uhg. Esso utilizza un protocollo denominato SCIP 2 il quale però è risultato non essere compatibile con la suite ARIA (non veniva riconosciuto come laser). Essa infatti è stata implementata per interfacciarsi a laser che comunichino mediante il protocollo SCIP 1. Si è allora proseguito il lavoro con un altro tipo di laser che utilizza appunto il protocollo SCIP 1. Le sue caratteristiche sono elencate di seguito.

Le caratteristiche del laser

Sono di seguito riportate le informazioni del laser utilizzato (Figura 5). Un esempio di rilevazione è rappresentato in Figura 6.

Figura 5: il laser

- Nome prodotto: Scanning laser range finder
- Modello: URG 04LX
- Marca: HOKUYO
- Protocollo di comunicazione: SCIP 1
- Capacità di rilevamento: da 20 a 5600 mm (maggiore affidabilità fra 60 e 4095 mm)
- Accuratezza di rilevamento: dai 20 ai 1000 mm c'è un margine di errore di 10 mm mentre dai 1000 ai 4000 c'è un margine di errore dell' 1% della misurazione
- Angolo di scansione: 240 gradi
- Risoluzione angolare: 0,36 gradi (360/1024)
- Interfaccia: USB

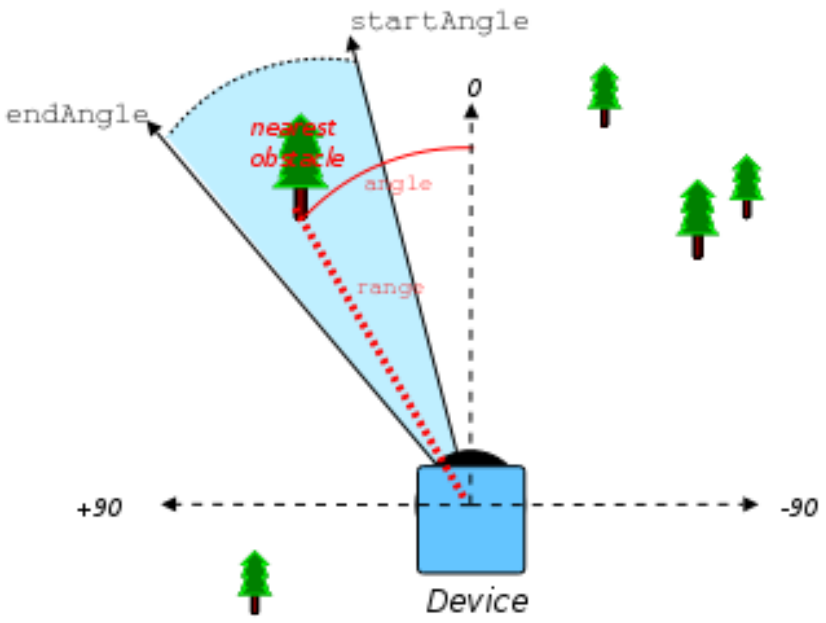

Figura 6: Rappresentazione schematica di una scansione laser

Le modifiche al programma “Caramelle”

Il programma “Caramelle” era già stato precedentemente realizzato ed il suo compito era quello di comandare il robot affinché circolasse in un ambiente qualsiasi evitando gli ostacoli e trasportando un contenitore di caramelle da distribuire (da qui appunto il nome del programma). La prima versione di questo programma era stata progettata per ricevere dati di rilevamento di ostacoli solamente dai sonar. Le modifiche effettuate sono quindi state necessarie per permettere al robot di ricevere i dati provenienti anche dal laser. All'inizio c'è la parte di “include” che fa riferimento sia all'Header file “Caramelle.h” (che contiene a sua volta un altro “include” per la libreria Aria) sia alla libreria String, necessaria per poter invocare metodi già pronti che lavorino su stringhe di caratteri. Dopo alcune dichiarazioni e assegnamenti di variabili necessarie all'inizializzazione del programma, c'è il metodo principale ossia il “main”, responsabile dell'avvio di tutto il programma. La prima cosa che esso fa è di creare delle istanze delle classi fondamentali: ArRobot, ArSonarDevice, ArBumpers. Esse sono responsabili rispettivamente di controllare i movimenti del robot, di ricevere i dati del sonar e infine di controllare i sensori del paraurti

collocato sulle pinze del robot . Il passo successivo consiste nel creare delle istanze delle action, di cui si è parlato anche prima.

```
ArActionStallRecover recoverAct;

ArActionBumpers bumpAct;

 ArActionAvoidFront avoidFront;

ArActionConstantVelocity constantVelocityAct("Constant Velocity", 1000);
```

Ne viene creata una per evitare gli ostacoli di fronte (“AvoidFront”) , per reagire a situazioni di stallo delle ruote (“recoverAct”), per reagire a imput provenienti dai paraurti (“bumpAct”) e per la velocità (“ConstantVelocityAct”). Quest'ultima parte del programma è stata modificata in quanto, dopo varie prove, si è visto che è sufficiente una sola istanza della classe atta ad evitare gli ostacoli frontali. Per “AvoidFront” si potrebbero anche specificare la distanza che deve avere l’ostacolo affinché possa essere considerato tale, la velocità che deve avere il robot durante l’azione e infine l’angolo di rotazione per evitare l’ostacolo. Non è però necessario fare queste operazione, in quanto i parametri che vengono caricati di default sono già funzionanti. Si è poi constatato che anche l’istanza “avoidSide” della classe atta ad evitare gli ostacoli laterali risulta superflua. Dopo aver inizializzato la libreria Aria, viene creato poi il connettore al robot. A differenza di quanto accadeva nella versione precedente, nella quale si utilizzava la classe simpleConnector, ora verrà utilizzata un'altra classe chiamata robotConnector, in quanto compatibile con il connettore al laser che viene creato subito dopo avere instanziato la classe ArLaserConnector. E’ necessario, infatti, passare un oggetto della classe ArRobotConnector e non della classe simpleConnector alla classe ArLaserConnector come parametro attuale al momento dell’instanziazione. Infatti è proprio la classe ArRobotConnector che si occupa dell’elaborazione dei parametri passati dalla linea di comando. Le righe di codice successive permettono al robot di ricevere dati di rilevamento sia dai sonar, sia dai paraurti. Questo grazie alla creazione di un oggetto “ArSonarDevice” e di uno “ArBumpers” e invocando poi il metodo “addRangeDevice” passandogli come parametro il loro indirizzo. Non serve aggiungere il laser come device in quanto viene fatto in automatico direttamente alla connessione. Da notare, anche, che è stata creata l’istanza della classe

“ArGripper” (“pinza”) che verrà poi utilizzata successivamente. Il passo seguente è quello di provare a connettersi al robot. In caso di insuccesso compare una scritta a video e il programma termina la sua esecuzione. Parser è l’istanza che rappresenta i parametri inseriti al momento dell’esecuzione, ovvero quelli che corrispondono agli argomenti “argv” e “argc” del metodo main. Se ad esempio è necessario usare un device per il robot diverso da quello standard, bisogna digitare al momento dell’esecuzione “-rp” seguito dal nome del device. Al connettore del robot bisognerà poi passare l’indirizzo di parser.

```
if (!robotConnector.connectRobot()){
  if (!parser.checkHelpAndWarnUnparsed()) {
    ArLog::log(ArLog::Terse, "Could not connect to robot, will not have parameter file so options displayed later may not include everything");
  }
  else {
    ArLog::log(ArLog::Terse, "Error, could not connect to robot.");
    Aria::logOptions();
    Aria::exit(1);
  }
}
```

La stessa cosa avviene per la connessione al laser.

```
if (!laserConnector.connectLasers(false, false, true))
{
  printf("Could not connect to lasers... exiting\n");
  Aria::exit(2);
}
```

Dopodiché vengono abilitati i motori del robot e si attribuisce il “peso”, ovvero il livello di importanza attribuito alle diverse azioni descritte precedentemente (rappresentato con un numero intero). Durante l’esecuzione del programma vengono quindi valutate in ogni istante tutte le azioni dando la priorità a quella con peso maggiore. L'ordine decrescente per importanza delle azioni è il seguente: bumpAct (100), recoverAct(90), avoidFront (79), constantVelocityAct (50). Il robot dunque tende ad accelerare e a raggiungere la velocità costante scelta solo quando non intervengono le altre action che hanno importanza maggiore.

```
tobor.addAction(&recoverAct, 90);

tobor.addAction(&bumpAct, 100);

 tobor.addAction(&avoidFront,79);

tobor.addAction(&constantVelocityAct, 50);
```

I pesi sono differenti rispetto alla versione precedente in quanto ci sono quattro action e non cinque. E’ stato quindi necessario, dopo diverse prove, ricalibrarli. Viene poi attivato l’ “ArRobot Task Cycle” (attraverso il comando “tobor.runAsync(true);”) che permette inoltre di stampare a video informazioni provenienti dai vari sensori installati sul robot. Con alternanza di 10 secondi si attiva in particolare la modalità laser (grazie a una precedente instanziazione della classe ArModeLaser), caratteristica non presente nella versione precedente. La parte finale del programma serve per dare input particolari al robot in caso di stallo causato dallo scontro con un ostacolo non visto. Ad esempio se esso si trova di fronte un ostacolo di dimensioni ridotte, il robot lo afferra con la pinza e tenta di trasportarlo oppure emette un suono. Nel caso in cui invece l’ostacolo impedisca al robot di procedere, viene attivata la retromarcia ed effettuata una rotazione. Questo dovrebbe permettere al robot di ricominciare a muoversi normalmente.

Nelle appendici 1 e 2 sono presenti nella loro interezza le due versioni del programma “Caramelle”, quella originale e quella modificata. Sono evidenziate le parti modificate e/o aggiunte.

Il file bash per avviare il programma

```
#!/bin/bash

./caramelle -rp /dev/ttyUSB0 -laserType urg -lp /dev/ttyACM0 -lpt serial -lds -90 -lde 90 -connectLaser
```

Per evitare di dover digitare ogni volta dalla riga di comando tutti i comandi necessari per far partire il programma “Caramelle” in maniera corretta, è stato creato un file bash in maniera tale da poter semplicemente digitare il suo nome per far partire tutto in automatico. In dettaglio i comandi inviati sono:

- “-rp”, che sta per “robot port”, seguito dal nome di una porta, serve a indicare al programma la porta a cui connettersi per poter comunicare col robot. In caso non gli venisse fornita, il programma proverebbe a connettersi al simulatore e, in caso di fallimento, alla porta seriale che, come detto precedentemente, non è presente sul calcolatore utilizzato. E’ quindi necessario fornire la porta seriale simulata di cui sopra.
- “-laserType” seguito dal tipo di laser, serve a fornire al programma il tipo di laser utilizzato (in questo caso si tratta di un laser urg);
- “-lp”, che sta per “laser port”, seguito dal nome di una porta, serve per indicare la porta a cui il laser è collegato;
- “-lpt”, che sta per “laser port type” seguito da un tipo di porta, serve a fornire al programma il tipo di porta utilizzato dal laser (in questo caso è di tipo seriale, anche se, come nel caso del robot, è in realtà una seriale simulata il cui driver era già presente sul calcolatore utilizzato);
- “-lds” e “–lde”, che stanno rispettivamente per “laser degrees start” e “laser degrees end” , seguiti da un angolo, servono per fornire al programma l'intervallo angolare da considerare nelle rilevazioni. Bisogna infatti tener presente che il laser utilizzato arriva a 240 gradi e di conseguenza potrebbe venire disturbato dai cavi che sono collocati dietro di esso. Per questo, attraverso questi due comandi, è opportuno impostare una visuale di 180 gradi al massimo (fra -90 e 90);

- “-connectLaser” è invece l’ultimo comando che fa sì che venga effettivamente instaurata la connessione col laser con i parametri appena discussi.

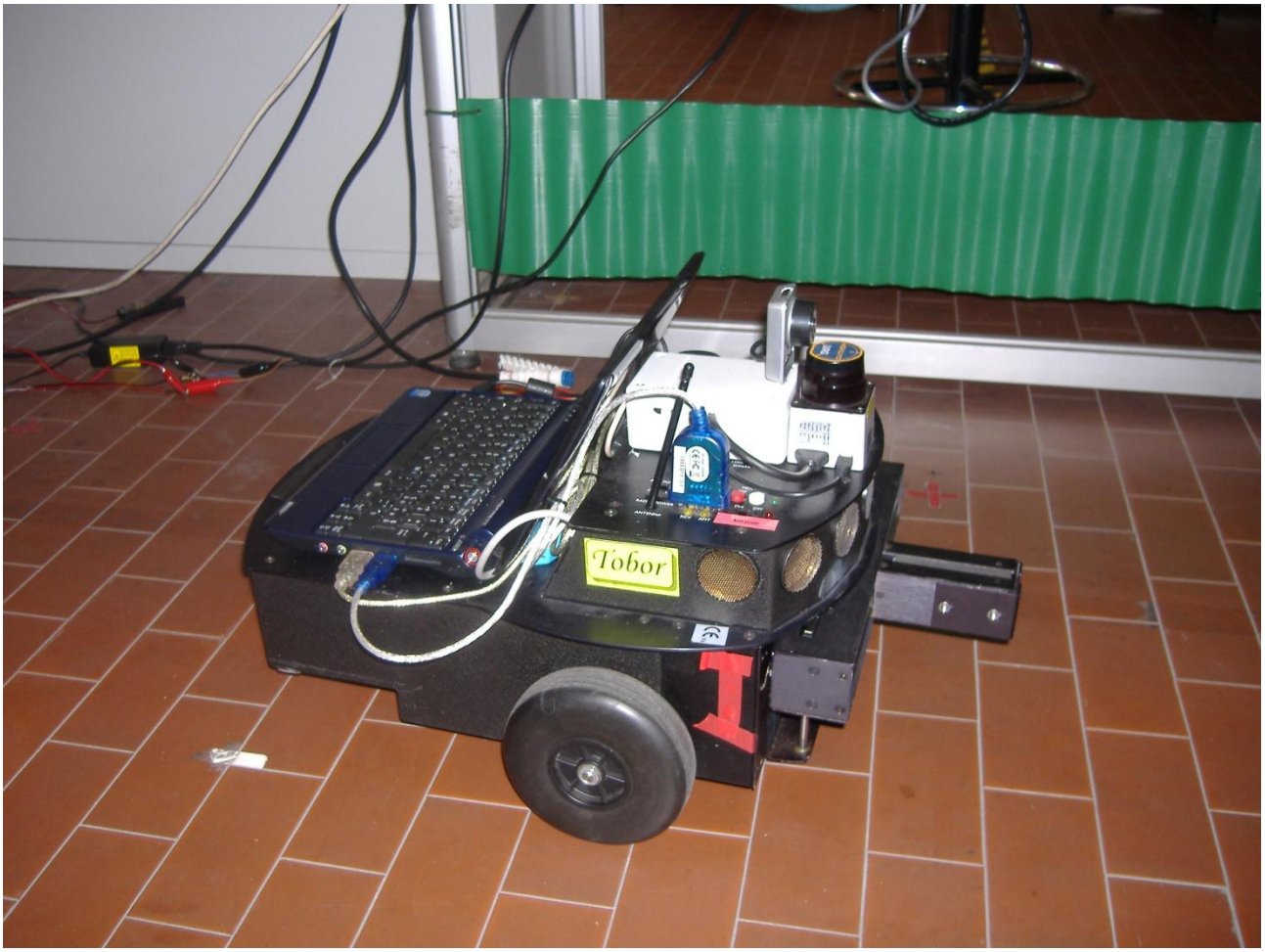

Figura 7: Tobor con calcolatore e laser

Capitolo 2

Il passo successivo: ARNL.

Introduzione

Si è pensato di continuare lo studio sul laser e su eventuali librerie e software che sfruttassero le sue potenzialità. Sempre sul sito della MobileRobots è disponibile un software chiamato “Arnl”. In particolare esistono due versioni: una basata sul sonar e l’altra basata invece sul laser. Il compito di questo software è quello di localizzarsi all’interno di un ambiente la cui mappa deve già esistere su file o essere creata al momento. Arnl è un programma “server”, ossia un programma la cui utilità sta nel fornire informazioni a un altro eventuale programma detto “client” che si connette ad esso.

Modifiche software del netbook

E’ stato innanzitutto necessario installare “BaseArnl” , indispensabile per il funzionamento di entrambe le versioni di Arnl. Si è poi installato il software Mapper3, la cui finalità verrà illustrata successivamente.

I primi tentativi con SonArnl

Si è pensato di iniziare a lavorare sulla versione del software più semplice, ovvero quella che sfrutta solo i sonar per il suo funzionamento. Essa è anche molto più scarna come funzionalità in quanto non permette la creazione di mappe in tempo reale. Infatti, per poter effettivamente funzionare, è necessario fornirle, all’avvio, una mappa attraverso il comando “-map” seguito dal nome della mappa. Quest’ultima può essere solo realizzata manualmente mediante il software Mapper3basic già indicato all’inizio della relazione. La prima anomalia riscontrata è il non

riconoscimento del laser. Infatti, avviando il programma con gli stessi comandi del programma “Caramelle”, quelli relativi al laser non vengono riconosciuti. Andando poi a vedere come è stato scritto il programma, si nota che non è presente il connettore al laser. Una volta aggiunto, il laser viene riconosciuto e il programma sonArnlServer può partire. Per controllare efficacemente ciò che fa questo programma server, serve un programma client che può girare sia sulla stessa macchina sia su un’altra. Quello più conveniente da utilizzare è MobileEyes, sempre fornito dalla MobileRobots, dotato di un’interfaccia grafica molto intuitiva e semplice. Esso può essere avviato sulla stessa macchina oppure, molto più efficacemente, da remoto. In quest'ultimo caso, la prima cosa che viene richiesta è l’indirizzo IP e la password del server a cui connettersi (in questo caso bisogna fornire l’indirizzo IP e la password del calcolatore su cui sta girando SonArnlServer). In figura 8 vi è una rappresentazione schematica dei vari elementi che entrano in gioco.

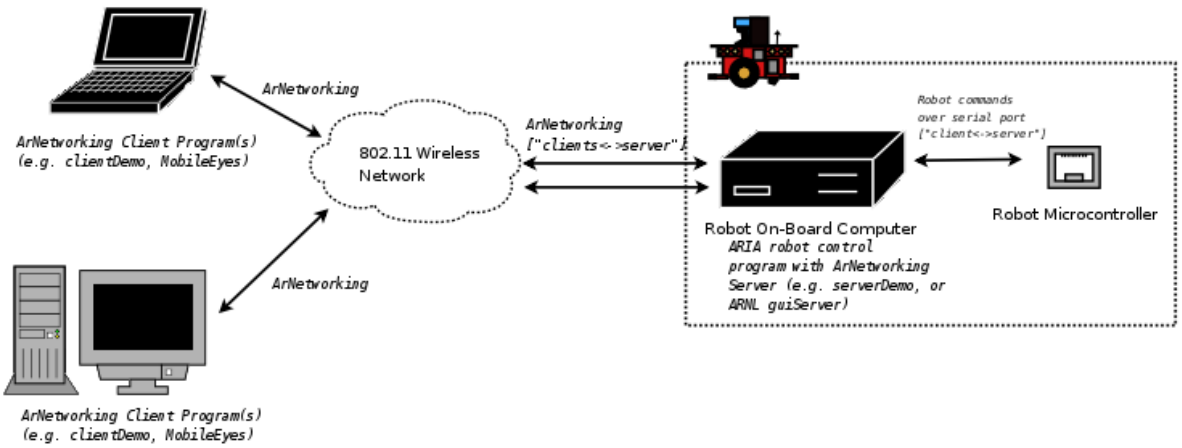

Figura 8: Rappresentazione schematica del client e del server con Arnl

Fornitegli queste informazioni, ciò che viene poi mostrato è la mappa al cui interno si trova il robot. Si nota subito che la posizione del robot sullo schermo non coincide assolutamente con quella effettiva nella stanza. E’ chiaro quindi che il robot non è riuscito a localizzarsi. I motivi sono fondamentalmente due. Innanzitutto la mappa, che è stata disegnata con Mapper3Basic, non può essere precisa al 100%; secondariamente i sonar sono strumenti inadeguati rispetto al laser che non viene utilizzato in questa versione nonostante venga riconosciuto. MobileEyes mette comunque a disposizione il comando di localizzazione manuale. E’ sufficiente infatti posizionare

col mouse il robot all'interno della mappa. Sullo schermo il laser è rappresentato attraverso dei puntini colorati e i sonar attraverso triangolini. E' inoltre possibile dare comandi di spostamento. Si riscontrano però altre anomalie. Funzionano infatti soltanto i comandi per sterzare a destra o a sinistra. Quelli per andare avanti o indietro non sembrano funzionare. Si scopre però che anch'essi hanno effetto nel momento in cui si disabilita l'impostazione "Safe Drive". La risoluzione di questo problema verrà spiegata successivamente. Un altro problema è il mancato funzionamento del comando "send robot" sempre presente in MobileEyes. Esso dovrebbe permettere di mandare il robot in qualunque parte della stanza semplicemente cliccando sulla mappa il punto desiderato. Mediante un calcolo di traiettoria ottimale, calcolo che viene effettuato dal server e non da MobileEyes (che è il client), il robot dovrebbe raggiungere in maniera completamente automatica ed autonoma il punto desiderato. Dopo un primo accenno di movimento, però, il robot si ferma e compare la scritta di errore. Anche la risoluzione di questo problema verrà spiegata successivamente.

ARNL con il laser

I tentativi successivi sono invece stati effettuati con l'altra versione di ARNL chiamata ArnlServer, quella che appunto usufruisce della precisione del laser. Il funzionamento è molto simile a SonArnlServer e lo sono anche i comandi per farlo partire. C'è però una fondamentale differenza: vi è la possibilità di disegnare la mappa della stanza in tempo reale. Al programma server stavolta non si passa nessuna mappa. In automatico viene quindi rilevato che si è in assenza di essa, e compaiono a video le istruzioni per tracciarla. Da MobileEyes (che è, come detto precedentemente, il programma client) è possibile infatti andare sul menu e cliccare sull'opzione di creazione di mappe. Dopo aver inserito il nome da dare al file che conterrà tutti i dati di scansione, viene data la possibilità di guidare il robot all'interno della stanza. Sarebbe opportuno effettuare questa operazione potendo osservare direttamente il robot, data la delicatezza e l'importanza di questa operazione. Più il robot esplora le varie aree della stanza, più la mappa sarà precisa. Mentre il robot naviga, il laser invia i dati al calcolatore il quale salva tutto sul file il cui nome era stato scelto all'inizio della scansione. Quando si pensa di aver effettuato abbastanza esplorazioni, si clicca sempre nel menu per interrompere il processo. Andando poi a vedere nella cartella che contiene il programma ArnlServer, si nota che è presente un nuovo file col nome

scelto e con estensione ".2d". Questo non è il file a tutti gli effetti della mappa, anche perché il programma per poter partire ha bisogno di un file con estensione ".map". Bisogna allora rielaborarlo con il software Mapper3, che non fa altro che aprire il file con estensione ".2d" e, dopo aver ripercorso passo dopo passo tutte le mosse del robot, creare il file con estensione ".map". A questo punto si puo' far ripartire il programma ArnlServer fornendogli la mappa appena creata. Con maggior probabilità, il robot stavolta riuscirà a localizzarsi. Anche in questo caso però si riscontrano gli stessi due problemi che si erano registrati anche con SonArnl: i comandi di "avanti" e "indietro" non funzionano con il "safe drive" attivato e il comando "send robot" non ha alcun effetto. Il difetto sta tutto nel file "pion1m.p" relativo ai parametri del modello del robot e che risiede nella directory "params" presente nella cartella di installazione di ARNL. Sono di seguito presentate (dopo le due immagini) le modifiche effettuate al file. Nelle appendici 3 e 4 sono invece riportate le due versioni del file, originale e modificata. Sono evidenziate le parti modificate e/o aggiunte.

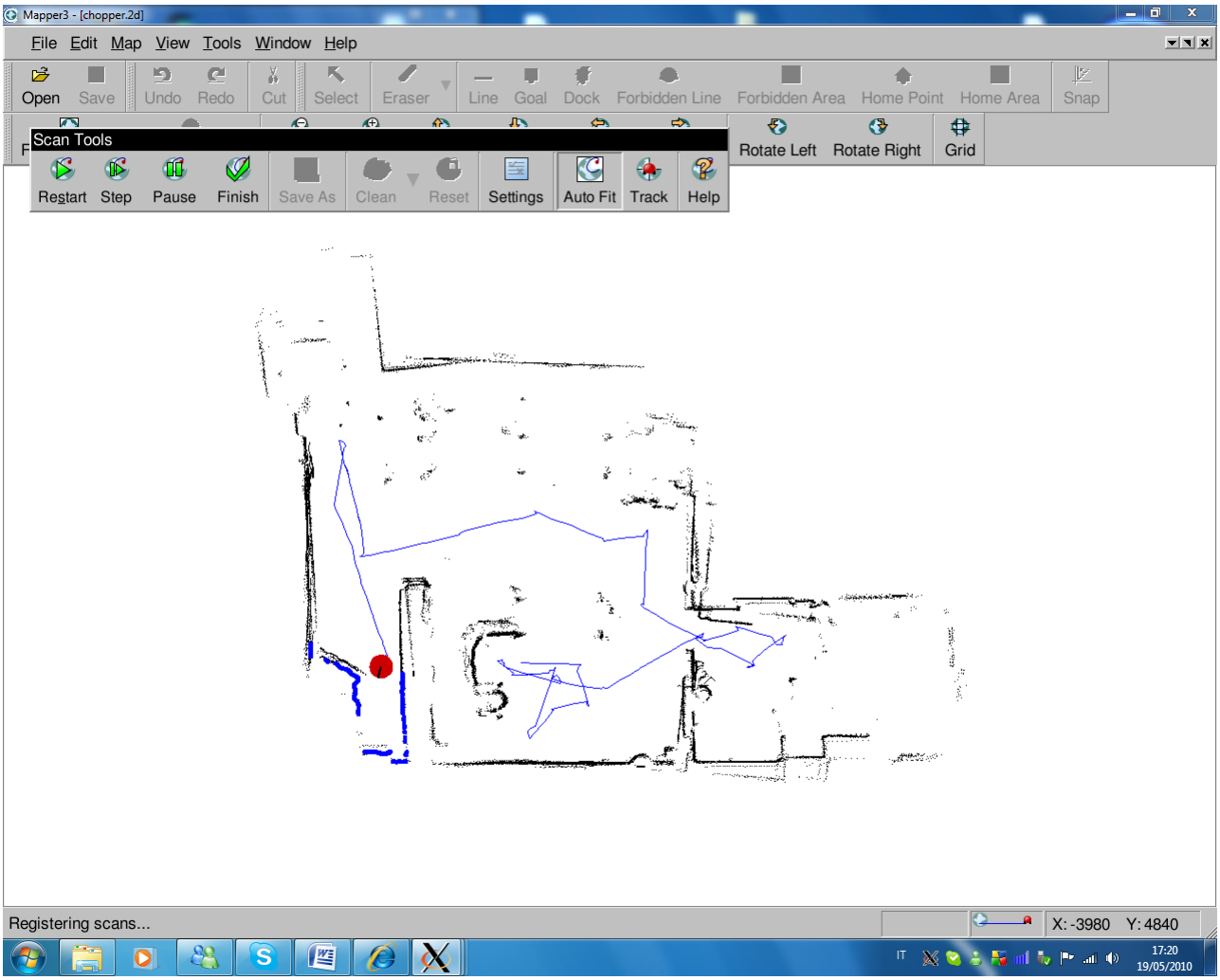

Figura 9: Il programa Mapper3 mentre ricostruisce la scansione effettuata dal laser di Tobor

In Figura 9 è possibile osservare il programma Mapper3 mentre rielabora tutti i movimenti fatti dal robot (rappresentato dal pallino rosso) durante la scansione dell'ambiente. La mappa della stanza nel frattempo viene disegnata.

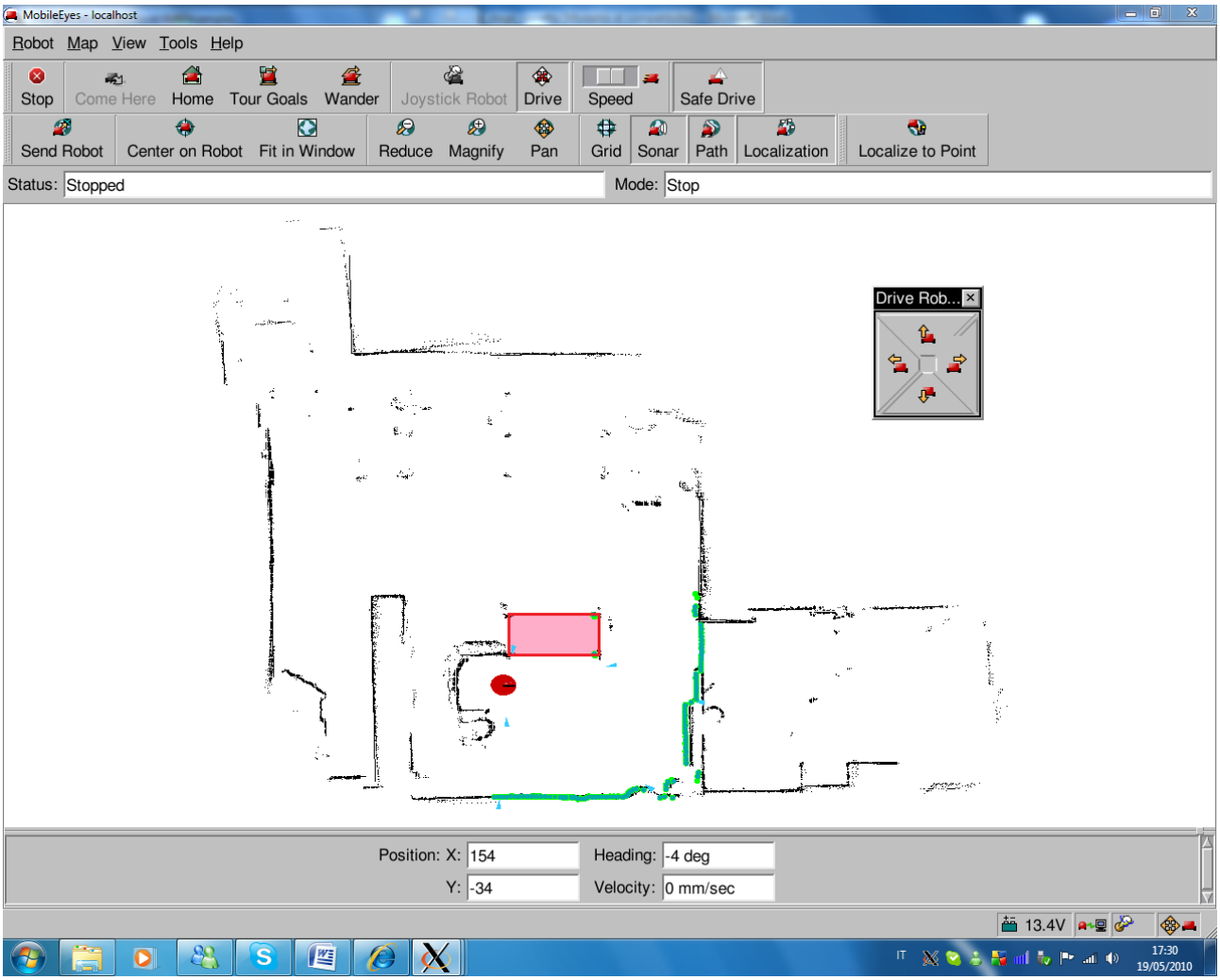

Figura 10: Il programma client MobileEyes

In figura 10 è invece possibile vedere il programma MobileEyes in azione. Il robot è perfettamente localizzato all'interno della mappa costruita con Mapper3. Se si osserva con attenzione la mappa, si nota che c'è una zona di colore rosso. E' una "forbidden area", ossia un'area vietata, aggiunta successivamente alla scansione in quanto in quell'area è presente un ostacolo che, essendo ad

un'altezza inferiore a quella in cui è posto il laser sul robot, non è stato visto. E'stato opportuno quindi modificare la mappa perché altrimenti il robot, nel calcolo della traiettoria ottimale dopo aver cliccato il comando "send robot", teneva conto di uno spazio libero che in realtà non era presente, e spesso questo lo faceva disorientare. I pallini azzurrini e verdi rappresentano ciò che vede il laser (che coincide perfettamente con le pareti, simbolo della localizzazione avvenuta) mentre i triangoli azzurri rappresentano i sonar.

Le modifiche al file "pion1m.p"

La prima modifica è consistita in un'aggiunta di un ottavo sonar, anche se non esiste in realtà.

```
;SonarNum 7            ; number of sonar on the robot

SonarNum8

SonarUnit 7 0 0 180
```

Questo perché anche se il numero dei sonar montati sul robot è 7, il firmware acquisisce i valori di tutti i sonar previsti (8) e li trasmette. Ciò dà luogo ad un frame extra, che causa una segnalazione di errore nel programma di interfaccia con il robot. Inoltre sono state modificate le velocità e le accelerazioni relative alle ruote del robot. Infatti nella versione precedente erano quasi tutte impostate al valore 0, il che, di fatto, era come affidare al robot il compito di stabilire a run-time i parametri adatti. L'operazione, però, non aveva successo perché venivano caricati parametri che risultavano errati e che, di conseguenza, non rendevano possibile il calcolo delle traiettorie con arnl. Modificando quindi manualmente i parametri a 1000, ad esempio, (in realtà qualunque numero abbastanza grande diverso da 0 va bene) si risolvono in un colpo solo quelle due famose anomalie discusse prima. Tutti gli altri parametri relativi a impostazioni del robot, dei sonar e del laser risultano corretti.

```
TransVelMax 1000       ; maximum desired translational velocity for the robot

RotVelMax 1000         ; maximum desired rotational velocity for the robot
```

```
;SettableAccsDecs false   ; if the accel and decel parameters can be set

SettableAccsDecs true

TransAccel 1000          ; translational acceleration

TransDecel 1000          ; translational deceleration

RotAccel 1000            ; rotational acceleration

RotDecel 1000            ; rotational deceleration
```

NOTA: il ";" sta a significare che tutto quello che segue nella stessa riga non viene considerato.

L'aggiunta della webcam e lo streaming video

Per permettere anche da remoto di poter manovrare il robot all'interno della stanza in maniera più efficiente, è stata installata su di esso una webcam Philips. Si è reso necessario di conseguenza installare un software video che potesse connettersi alla telecamera e che potesse essere invocato senza creare problemi anche da remoto. Si è optato per Luvcview. Digitando su terminale "luvcview", infatti, si apre una piccola finestra in cui è visualizzato ciò che vede la webcam.

Per integrare lo streaming video nell'interfaccia web, di cui si parlerà nel capitolo 4, si è scelto di utilizzare il programma mjpg-streamer, il cui compito è quello di catturare lo streaming video e renderlo disponibile su una porta scelta. E' sufficiente poi inserire nell'interfaccia web una particolare applet, chiamata Cambozola, con determinati parametri che indichino l'indirizzo del server con la relativa porta su cui mjpg-streamer invia lo streaming.

Capitolo 3

La stazione di ricarica

Introduzione

Come prosecuzione del lavoro, si è scelto di costruire una stazione di ricarica per il robot. La sua utilità sarà quella di ricaricare sia il robot, sia il calcolatore montato su di esso. Questa scelta va di pari passo con la creazione di un'interfaccia web per il comando a distanza del robot, argomento trattato nel capitolo 4.

La struttura della stazione di ricarica

Il primo problema affrontato è stato la scelta di come costruire materialmente la stazione. Ci si è ispirati a quella di un altro robot dell'ente ospitante. I due robot infatti sono quasi identici, tranne per una pinza che Tobor possiede nella parte anteriore, caratteristica mancante all'altro. Questa differenza è stata determinante. Infatti se l'altro robot può entrare dentro la sua stazione nella direzione di marcia, Tobor dovrà farlo in retromarcia. Si sono inoltre ipotizzati diversi sistemi di incanalamento del robot. E' sorto pertanto il problema della presenza di un ruotino posteriore che, in alcuni casi, faceva deviare la traiettoria del robot. Questo perché, all'inizio della retromarcia, dovendo ruotare sul proprio asse verticale, il ruotino si comporta come un timone, provocando spesso delle oscillazioni del robot. Dopo diversi tentativi, si è risolto il problema mettendo a punto la struttura evidenziata nelle figure 11 e 12.

Figura 11: Una fotografia della stazione di ricarica durante la costruzione

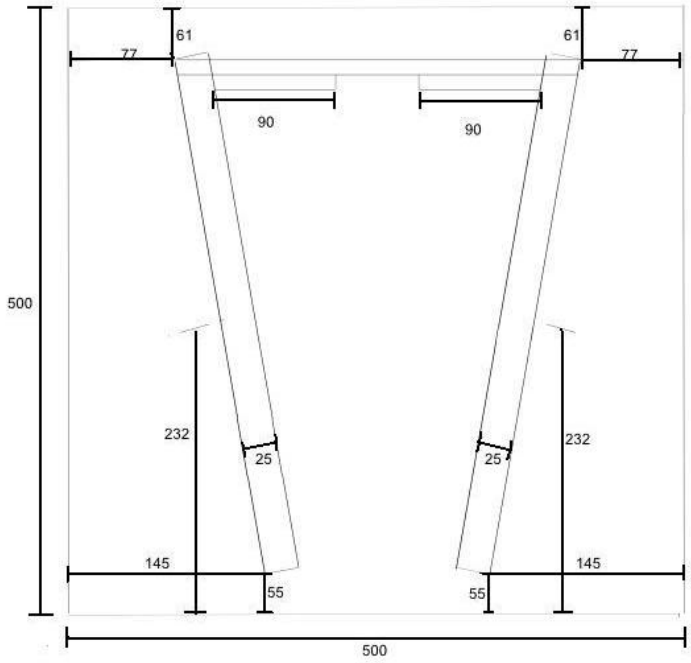

Figura 12: Rappresentazione schematica delle guide della stazione di ricarica per Tobor

Come si può notare, la struttura di incanalamento prevede due guide che servono a convogliare in maniera corretta le due ruote motrici di Tobor. Quando viene innestata la retromarcia possono verificarsi tre casi: il robot si trova posizionato perfettamente davanti alla stazione, il robot si trova leggermente spostato rispetto all'asse della stazione, il robot si trova davanti alla stazione ma leggermente inclinato rispetto all'asse. E' in questi ultimi due casi che risultano fondamentali le due guide: in base all'entrata del robot, la guida di destra o di sinistra farà in modo che il robot

raddrizzi la sua traiettoria per far sì che si agganci ai contatti in maniera regolare. Sono inoltre presenti due squadrette metalliche necessarie al bloccaggio delle ruote motrici, per evitare che possano scavalcare le guide, a causa di una velocità che non può comunque essere bassa, vista la necessità di una certa potenza che permetta al robot di agganciarsi ai contatti elettrici (dei quali si parlerà tra poco). Se la velocità fosse troppo bassa infatti, non solo ci sarebbero problemi di aggancio ai contatti, ma anche problemi di incanalamento, in quanto le ruote di gomma non avrebbero abbastanza forza per entrare correttamente nell'area di ricarica seguendo le guide. Non è stato invece necessario aggiungere nessun sistema di bloccaggio o incanalamento per il ruotino posteriore. La struttura appena descritta è risultata una soluzione ottimale per il rientro in retromarcia del robot.

I contatti elettrici

Per caricare sia la batteria del robot, sia quella del calcolatore, è stato necessario montare dei contatti elettrici sia sul robot, sia sulla stazione di ricarica. Per i primi si sono utilizzate due barrette di materiale isolante (una per i contatti della batteria del robot, una per quella del calcolatore), montate ai lati del ruotino posteriore, come in Figura 13.

Figura 13: I contatti elettrici di Tobor

La scelta del materiale isolante per le due barrette è stata necessaria in quanto il robot è fatto di alluminio, che è a sua volta un conduttore. Sulle due barrette è stato applicato del nastro adesivo di rame: due strisce (polo negativo e polo positivo) per la batteria del robot, due strisce per quella del calcolatore. Sono quindi presenti in totale quattro contatti. Se ne potevano realizzare anche soltanto tre, ma questo avrebbe implicato che i circuiti del robot non sarebbero stati separati da quelli del calcolatore. Come conseguenze, ci sarebbero potuti essere problemi nel momento dell'entrata in stazione e si sarebbero potuti creare anelli di massa.

I contatti elettrici sulla stazione di ricarica sono stati invece realizzati mediante quattro linguette metalliche (due per la batteria del robot, due per la batteria del calcolatore), montate su una barretta di materiale plastico, a sua volta posizionata al termine delle due guide descritte precedentemente, come mostrato in Figura 14.

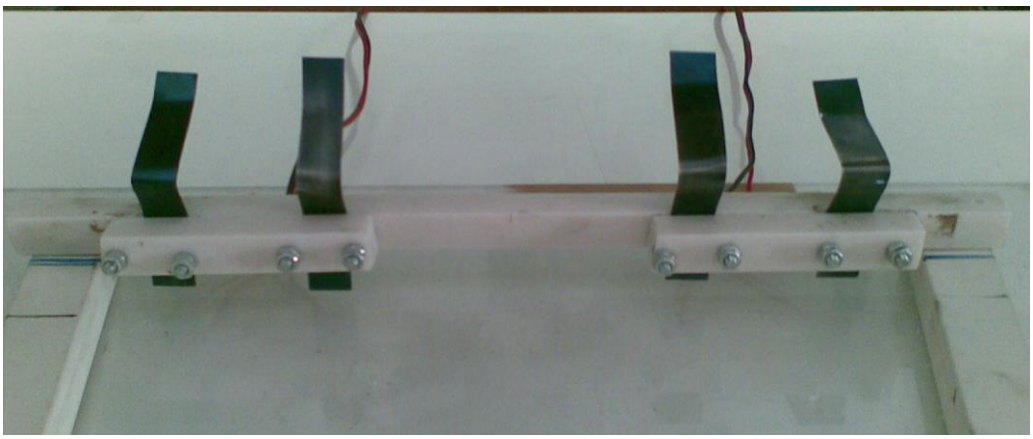

Figura 14: I contatti elettrici della stazione di ricarica

La forma delle linguette metalliche contribuisce inoltre al blocco del robot, al momento del suo rientro. Infatti, a causa dell'elasticità delle ruote motrici fatte in gomma, il robot tenderebbe ad avere un balzo in avanti, nel momento cui le ruote vengono bloccate dalle squadrette.

Capitolo 4

L’interfaccia web

Introduzione

Per completare il lavoro si è scelto di creare un’interfaccia web per comandare il robot a distanza, semplicemente attraverso l’uso di un browser, senza quindi la necessità di connettersi via ssh.

L’interfaccia: i link e i pulsanti

Per costruire la pagina web per Tobor, ci si è ispirati a quelle degli altri due robot presenti nell’ente ospitante. Di seguito è riportata l’immagine della pagine web per poter comandare il robot mediante un qualunque browser (Figura 15).

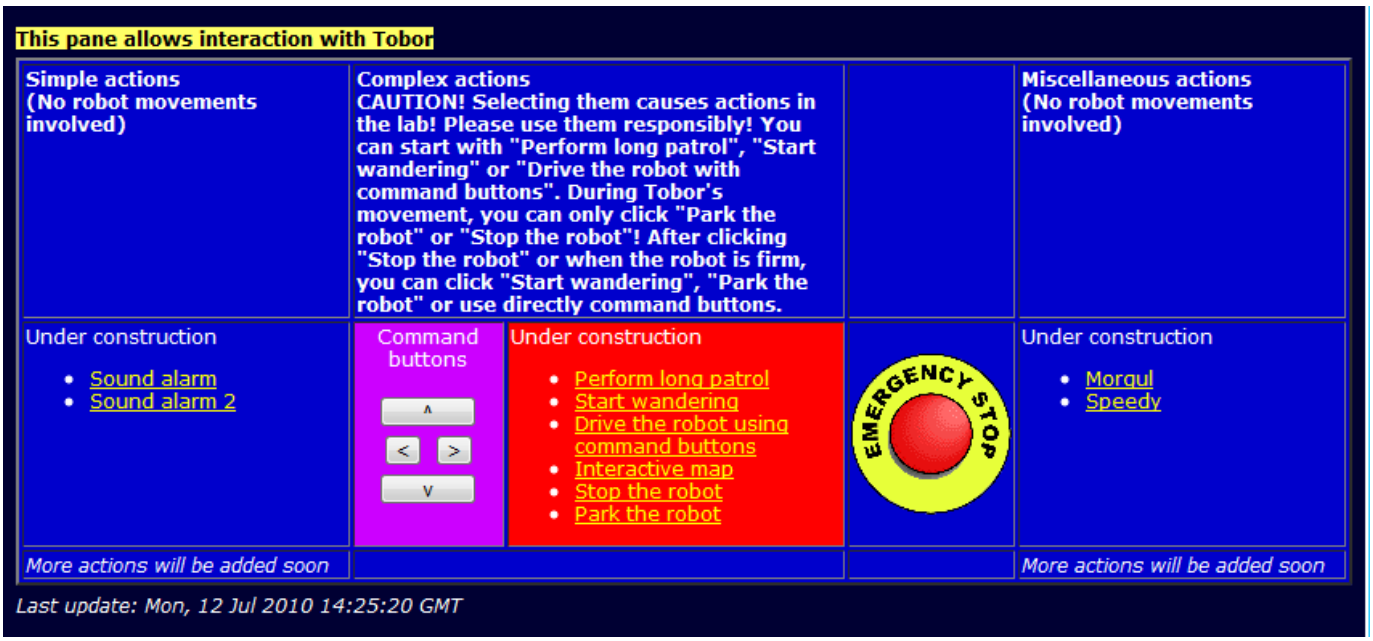

Figura 15: L'interfaccia web di Tobor

Nel riquadro rosso sono presenti sei diverse funzionalità rappresentate appunto da sei link. Facendo click su uno di essi, si attivano le rispettive modalità di navigazione del robot. Per evitare conflitti e malfunzionamenti, i sei link non possono essere premuti tutti in qualunque momento. All'inizio è possibile partire con una delle prime tre modalità: "Perform long patrol", "Start wandering", "Drive the robot using command buttons". Mentre il robot si muove è invece possibile solamente cliccare su "Stop the robot" o su "Park the robot". Quando è fermo è possibile cliccare su "Start wandering", "Park the robot", "Interactive map" o usare direttamente i comandi di "avanti, indietro, destra e sinistra". In qualunque momento, in caso di emergenza, si può cliccare su "EMERGENCY STOP".

L'implementazione con le CGI

Per prima cosa è stato necessario installare un servizio server http sul calcolatore, in maniera tale che ci si potesse connettere via http, visto che la pagina web risiederà su un altro calcolatore, e rendere disponibili in scrittura le porte USB anche per comandi esterni. Il primo problema da affrontare è stato su come implementare il meccanismo che permettesse di far eseguire programmi sul calcolatore montato su Tobor mediante appunto l'interfaccia web. C'erano almeno due possibilità: o attraverso le RPC (remote procedure call), modalità utilizzata per gli altri due robot e utile appunto per far eseguire programmi su un altro calcolatore connesso alla rete, o attraverso le CGI (common gateway interface). Si è optato per le CGI perché più rapide da realizzare. Queste ultime servono in realtà per la creazione di pagine web dinamiche e risultano utili nel caso in cui il server debba fornire all'utente una pagina web al momento, in base a certi parametri ricevuti. Si è però scelto di "aggirare" la vera utilità delle CGI, per ottenere comunque gli stessi risultati che si sarebbero avuti utilizzando le RPC. Facendo uso della shell Bash è possibile creare file con estensione .cgi che all'interno contengono sia il codice html per fare in modo che la CGI sia eseguita, sia comandi veri e propri per far eseguire programmi al server stesso. Siccome non serve nemmeno che vengano create pagine web, ma solo che vengano eseguiti programmi, le uniche righe di codice per il browser da inserire nello script Bash sono le seguenti:

```
echo "Status: 204 No Change"

echo

echo
```

Questo fa sì che il browser non apra in automatico un' altra pagina web, cancellando quella dei link, ma esegua soltanto la CGI, dentro la quale appunto saranno presenti i comandi. Gli eventuali output di testo prodotti dai programmi invocati, è necessario vengano indirizzati al device "null" affinchè non creino problemi all'esecuzione della CGI. E' di seguito riportato un esempio:

```
/usr/local/Arnl/examples/goalClientHome >/dev/null
```

I link e i pulsanti in dettaglio

- "Perform long patrol": il robot esegue un percorso prefissato. Nella relativa CGI ("tourGoals.cgi") viene dapprima invocato un programma che fa percorrere al robot circa mezzo metro affinché possa portarsi all'esterno della stazione di ricarica. Poi viene invocato il programma "arnlServer" (presentato nel capitolo 2), a cui viene passata una mappa del laboratorio con evidenziati dei punti specifici della stanza (compresa l'esatta posizione che il robot dovrà avere in quel punto), attraverso i quali il robot dovrà passare. Questo grazie anche all'invocazione di una serie di programmi client che invieranno al programma server (sempre "arnlServer") i comandi necessari. La struttura di ogni programma client è basata semplicemente sull'invio del seguente comando:

  ```
  client.requestOnceWithString("gotoGoal", "1");
  ```

  Il numero "1" sta a significare quindi che il robot dovrà andare a posizionarsi nel punto (goal) chiamato appunto "1" all'interno della stanza. Discorso analogo vale dunque anche per tutti gli altri 8 punti, di cui l'ultimo è chiamato "0", che è l'ultimo punto in cui si posizionerà il robot prima dell' uccisione del programma server (mediante il comando "killall -9") e dell'invocazione dell'ultimo programma che farà compiere al robot una rotazione di 180 gradi e che poi farà entrare il robot in stazione in retromarcia. Il punto "Home" è invece quello di partenza. Di seguito è riportata la mappa della stanza con i punti evidenziati (Figura 16). Si puo' notare che il goal "0" è leggermente spostato dall'asse della

stazione di ricarica. Questo è stato necessario per correggere quel margine di errore del laser, le cui rilevazioni tendevano a far posizionare il robot leggermente spostato rispetto alla posizione ottimale per il rientro.

Figura 16: La mappa dell'ente ospitante

Nell'appendice 5 è riportata la CGI relativa a questa modalità, mentre nell'appendice 6 è riportato il programma "goalClient1", uno degli otto programmi client invocati in serie, che differiscono solo per il nome del goal. Quando un programma client fallisce oppure riceve dal server l'informazione di avvenuto raggiungimento del goal desiderato, termina la sua esecuzione. La CGI può quindi invocare il successivo.

- “Start wandering”: il robot esegue una modalità di navigazione analoga a quella del programma “Caramelle” di cui si è parlato nel primo capitolo. Siccome questa modalità si può invocare sia all’inizio, sia durante i movimenti del robot, nella relativa CGI (“wander.cgi”) c’è prima un tentativo di invocazione del programma client “wander”, che invia il comando “client.requestOnce("wander");” al server (se è già attivo); in caso contrario viene invocata una versione del programma “arnlServer” già predisposta per quella modalità (è stata infatti aggiunta la riga di codice “modeWander.activate();” in fondo al programma). Dopo un certo numero di secondi (scanditi con il comando “sleep”) la CGI fa rientrare il robot nella stazione di ricarica. Questo mediante l’invocazione del client che invia l’input al server di raggiungere il goal “0” e la successiva invocazione del programma che fa compiere al robot una rotazione di 180 gradi e che fa entrare il robot nella stazione in retromarcia. Nell’appendice 7 è riportata la CGI relativa a questa modalità.

- “Drive the robot using command buttons”: il robot può essere pilotato mediante i pulsanti del riquadro di sfondo rosa. Viene invocato un programma server già predisposto alla modalità “drive” (grazie alla riga di codice aggiunta “modeRatioDrive.activate();”). Dopodichè è possibile schiacciare i pulsanti di guida: avanti, indietro, destra, sinistra. Viene quindi invocato il programma client relativo, molto simile nella struttura agli altri, tranne che per le righe di codice di base che permettono la creazione di un pacchetto da inviare poi al server. Dentro di esso dovranno esserci le velocità desiderate da imprimere al movimento delle ruote. Nel caso del client per mandare avanti il robot, il codice necessario è il seguente:

```
ArNetPacket packet;
packet.doubleToBuf(100);
packet.doubleToBuf(0);
packet.doubleToBuf(50); // use half of the robot's maximum.
```

```
packet.doubleToBuf(0);

client.requestOnce("ratioDrive", &packet);
```

- “Interactive map”: viene aperta una nuova pagina web in cui è raffigurata la mappa del laboratorio con evidenziati i vari Goals. Cliccando su uno di questi ultimi, il robot si posizionerà su di esso. Questo grazie all’invocazione di una CGI che, a sua volta, invoca il programma client relativo a quel goal, come descritto precedentemente.

- “Stop the robot”: il robot si ferma. Questo avviene grazie all’invocazione da parte della relativa CGI (“stop.cgi”) di un programma client che contiene la riga di codice “client.requestOnce("stop");”, comando che permette al server di bloccare le ruote. Segue poi l’uccisione di eventuali altri programmi client attivi, mediante il comando “killall -9” seguito dal nome del programma. Nell’appendice 8 è riportata la CGI relativa a questa modalità

- “Park the robot”: il robot si posiziona prima sul goal “0”, compie una rotazione di 180 gradi e poi rientra in stazione in retromarcia. Questo sempre grazie all’invocazione da parte della relativa CGI (“parkRobot.cgi”) del programma client per il goal “0” e del successivo programma per far compiere al robot una rotazione di 180 gradi e per far rientrare il robot in retromarcia.

- “Emergency stop”: il robot interrompe qualunque attività. Vengono disattivati tutti i programmi in esecuzione, sia client, sia server, sempre mediante il comando “killall -9”.

Durante l’invocazione di tutte le CGI, quindi, la struttura ricorrente dei programmi invocati vede sempre un programma server (“arnlServer”) che ha il compito fondamentale di tenere registrati i dati di localizzazione (dove si trova il robot all’interno della stanza) e di inviare i comandi meccanici alle ruote motrici del robot mediante la porta USB, e un programma client che ha invece il compito

di inviare al server i comandi necessari a far fare al robot ciò che l'utente desidera (ciò che viene cliccato sulla pagina web).

I problemi nel rientro in stazione

Durante le varie prove effettuate, l'unico problema che, in alcuni casi, si è manifestato, è stato l'errato, o il mancato rientro di Tobor in stazione. I motivi del verificarsi di questo fenomeno sono due. Il primo è che il laser non fa delle rilevazioni precise al 100%; il secondo è che, per come è fatta la struttura della stazione di ricarica, il margine accettabile di errore di posizionamento davanti ad essa, è molto basso. Come già detto precedentemente è stato innanzitutto necessario spostare il goal "0" per correggere almeno in parte l'errore del laser. Si è dovuta poi aggiungere nelle CGI in cui sono contenuti i comandi per il rientro una routine che intervenga nel caso ci sia un rientro errato o mancato (questo lo si capisce sempre grazie al comando "acpi-a"). Essa riinvoca il programma che fa compiere un metro in avanti al robot e riattiva arnlServer in modalità di wandering per 15 secondi. Dopodichè vengono riinvocati i programmi per il rientro. Tutto questo per cinque volte. Si è visto che nella maggior parte dei casi il robot riesce alla fine ad entrare regolarmente. Il rischio, anche se basso, di un mancato rientro è comunque presente. Come ulteriore ausilio al robot nell'individuare la stazione di ricarica, è stato aggiunto del cartone arancione nella parte posteriore della stazione di ricarica (Figura 17).

Figura 17: La stazione di ricarica ultimata

La pagina di stato

A fianco della pagina di comando di Tobor, viene caricata un'altra pagina web che mostra lo stato del robot. Viene visualizzata in tempo reale ogni mossa di Tobor. Ad esempio compare il prossimo goal a cui il robot è diretto, oppure se si trova in modalità di wandering, o altre informazioni sullo stato di Tobor. Questo grazie all'implementazione di una CGI che, a differenza dei casi precedenti, carica una vera e propria pagina web. Infatti nello script Bash è presente la riga di codice "echo -e "Content-type: text/html\n". Per visualizzare poi le varie caratteristiche di stato, vengono invocati dei client che stampano a video le informazioni richieste, filtrate, dove necessario, attraverso il comando "grep". Ad esempio, per sapere se il robot è entrato in stazione correttamente, la CGI invoca il comando "acpi –a" che restituisce una serie di informazioni sulla batteria del calcolatore di Tobor e, mediante appunto il "grep", vede se è presente la parola "off-line". In quel caso stampa a video "Robot not docked". E' di seguito riportata un'immagine della pagina di stato nella situazione di robot spento e correttamente entrato in stazione (Figura 18). Nell'appendice 9 è invece riportato il codice della CGI della pagina di stato.

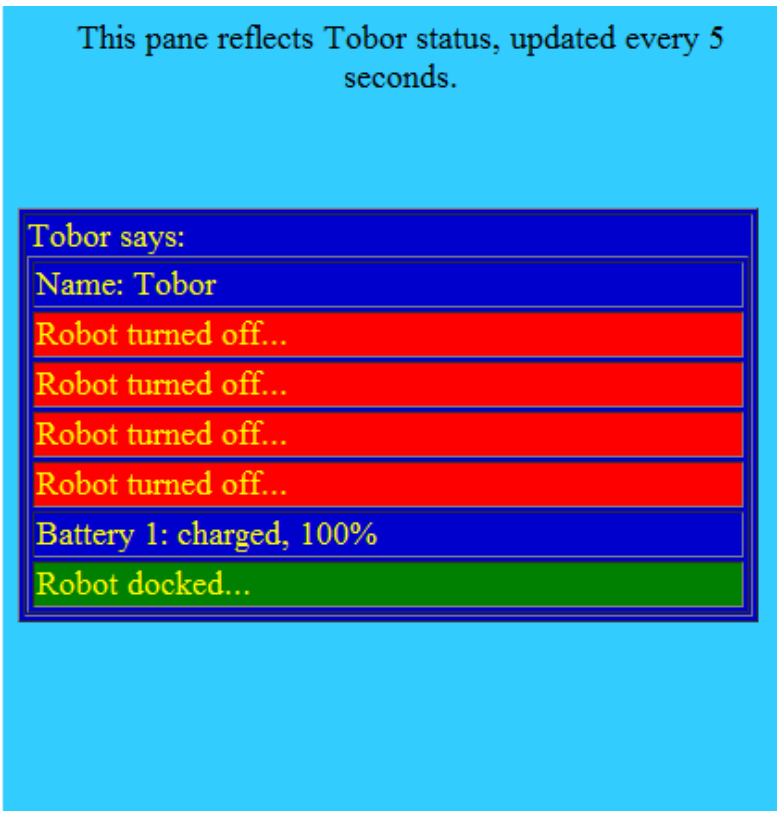

Figura 18: La pagina di stato

Problemi ancora aperti, conclusioni e sviluppi futuri

Alla fine del lavoro, i risultati ottenuti sono stati soddisfacenti in relazione agli obiettivi che ci si era prefissati. Si è partiti da un robot privo di laser e di calcolatore, comandabile solo via radio da un calcolatore interno all'ente ospitante. Si è arrivati a un robot dotato di calcolatore e di laser, capace di ricevere comandi da rete mediante browser e di localizzarsi e di muoversi all'interno di mappe di ambiente. Ciò che rimane ancora irrisolto è quel margine di possibilità di non rientro in stazione di Tobor, problema a cui, senza l'utilizzo di altri strumenti, non esiste soluzione in quanto, come già detto, l'imprecisione del laser non è completamente correggibile. Come prosecuzione del lavoro, si potrebbe lavorare all'implementazione di un meccanismo più efficace di rientro, simile a quello degli altri due robot dell'ente ospitante. Si tratterebbe di installare un programma sul calcolatore di Tobor in grado di rilevare la presenza di una fonte luminosa, collocata in una posizione opportuna sulla stazione di ricarica, attraverso l'utilizzo della webcam. In base alla posizione della fonte luminosa, il programma dovrebbe calcolare la traiettoria ottimale di rientro.

Figura 19: Tobor nelle vicinanze della stazione di ricarica

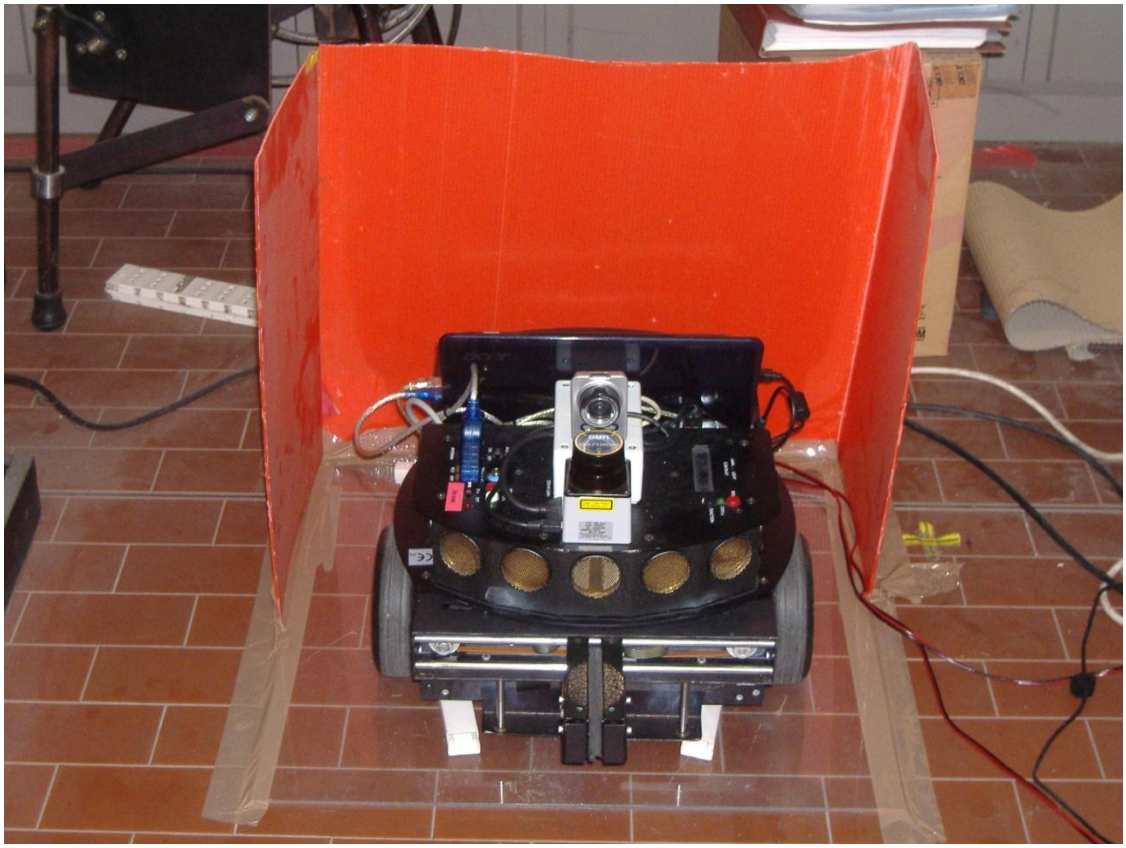

Figura 20: Tobor all'interno della stazione di ricarica

Appendice 1

Il programma “Caramelle.cpp” prima delle modifiche

```
#include "caramelle.h"

#include <string>


/* function to display a byte as a string of 8 '1' and '0' characters. */

std::string byte_as_bitstring(char byte)

{

	char tmp[9];

	int bit;

	int ch;

	for(bit = 7, ch = 0; bit >= 0; bit--,ch++)

		tmp[ch] = ((byte>>bit)&1) ? '1' : '0';

	tmp[8] = 0;

	return std::string(tmp);

}


/* function to display a 2-byte int as a string of 16 '1' and '0' characters. */

std::string int_as_bitstring(ArTypes::Byte2 n)

{

	char tmp[17];

	int bit;

	int ch;

	for(bit = 15, ch = 0; bit >= 0; bit--, ch++)
```

```
                tmp[ch] = ((n>>bit)&1) ? '1' : '0';
        tmp[16] = 0;
        return std::string(tmp);
}

/* Some events might only be detectable in one robot cycle, not over the
* 1-second period that the main thread sleeps. This cycle callback will detect
* those and save them in some global variables. */
bool wasLeftMotorStalled = false;
bool wasRightMotorStalled = false;
ArTypes::UByte2 cumulativeStallVal = 0;
ArTypes::UByte2 cumulativeRobotFlags = 0;
bool wasLeftIRTriggered = false;
bool wasRightIRTriggered = false;
bool wasEStopTriggered = false;

bool cycleCallback(ArRobot* tobor)
{
        cumulativeStallVal |= tobor->getStallValue();
        wasLeftMotorStalled = wasLeftMotorStalled || tobor->isLeftMotorStalled();
        wasRightMotorStalled = wasRightMotorStalled || tobor->isRightMotorStalled();
        wasEStopTriggered = wasEStopTriggered || tobor->getEstop();
        wasLeftIRTriggered = wasLeftIRTriggered || (tobor->hasTableSensingIR() && tobor-
>isLeftTableSensingIRTriggered());
        wasRightIRTriggered = wasRightIRTriggered || (tobor->hasTableSensingIR() && tobor-
>isRightTableSensingIRTriggered());
```

```
	return true;
}


/* main function */
int main(int argc, char **argv)
{
// robot and devices
	ArRobot tobor;
	ArSonarDevice sonar;
	ArBumpers bumpers;
	ArIRs ir;

// the actions we'll use to wander and avoid obstacles
	ArActionStallRecover recoverAct;
	ArActionAvoidSide avoidSide;
	ArActionBumpers bumpAct;
	ArActionAvoidFront avoidFrontNearAct("Avoid Front Near", 400, 0, 90);
	ArActionAvoidFront avoidFrontFarAct("Avoid Front Far", 600, 100, 15);
	ArActionConstantVelocity constantVelocityAct("Constant Velocity", 100);

// initialize aria and aria's logging destination and level
	Aria::init();
	ArLog::init(ArLog::StdErr, ArLog::Normal);
```

```
// connector
        ArSimpleConnector connector(&argc, argv);
        if (!connector.parseArgs() || argc > 1)
        {
                connector.logOptions();
                exit(1);
        }

        printf("This program will make the robot wander around, avoiding obstacles, and print some data and events.\nPress Ctrl-C to exit.\n");

// add the range devices to the robot
        tobor.addRangeDevice(&sonar);
        tobor.addRangeDevice(&bumpers);
        tobor.addRangeDevice(&ir);

        ArGripper pinza=ArGripper(&tobor);

// try to connect, if we fail exit
        if (!connector.connectRobot(&tobor))
        {
                printf("Could not connect to robot... exiting\n");
                Aria::shutdown();
                return 1;
        }
```

```
// turn on the motors, turn off amigobot sound effects (for old h8-model amigobots)
        tobor.enableMotors();
//      tobor.comInt(ArCommands::SOUNDTOG, 0);

// add the actions created above
        tobor.addAction(&recoverAct, 100);
        tobor.addAction(&bumpAct, 75);
        tobor.addAction(&avoidFrontNearAct, 50);
        tobor.addAction(&avoidFrontFarAct, 49);
        tobor.addAction(&avoidSide,48);
        tobor.addAction(&constantVelocityAct, 25);

// Cycle callback to check for events
        tobor.addUserTask("checkevents", 1, new ArGlobalRetFunctor1<bool, ArRobot*>(&cycleCallback, &tobor));


// start the robot running, true means that if we lose robot connection the
// ArRobot runloop stops
        tobor.runAsync(true);
        tobor.lock();
        printf ("Il tipo della pinza e' %d\n\n",pinza.getType() );
        tobor.unlock();

        tobor.lock();
```

```
        pinza.liftDown(); //Apre la pinza
        tobor.unlock();


        char pinzagiu=true;
//ArUtil::sleep(10000);









// Print data header
#define HEADFORMAT "%-24s %-5s %-16s %-5s %-6s %-6s %-16s %-8s %-8s %-8s %-8s %-8s %-8s %-10s %-10s %-5s %-5s %s"
#define DATAFORMAT "%-24s %03.02f %-16s %-5s %-6s %-6s %-16s %-8d %-8d %-8g %-8g %-8s %-8s %-10lu %-10lu %-5s %-5s" // doesn't include bumps details on end
        printf("\n" HEADFORMAT "\n\n",
            "Time",
            "Volts",
            "Flags",
            "EStop",
            "StallL",
            "StallR",
            "StallVal",
            "#SIP/sec",
            "#Son/sec",
            "Vel L",
```

```
                "Vel R",
                "DigIns",
                "DigOuts",
                "Enc L",
                "Enc R",
                "IR L",
                "IR R",
                "Cur Bumps, (Last Bump Pose)"
               );

// Request that we will want encoder data

        tobor.requestEncoderPackets();



// Print data every second
        char timestamp[24];
        while(tobor.isRunning()) {
                tobor.lock();
                time_t t = time(NULL);
                strftime(timestamp, 24, "%Y-%m-%d %H:%M:%S", localtime(&t));
                printf( DATAFORMAT,
                        timestamp,
                        tobor.getRealBatteryVoltage(),
                        int_as_bitstring(cumulativeRobotFlags).c_str(),
```

```
            (wasEStopTriggered ? "YES" : "   "),
            (wasLeftMotorStalled?"YES":"   "),
            (wasRightMotorStalled?"YES":"   "),
            int_as_bitstring(cumulativeStallVal).c_str(),
            tobor.getMotorPacCount(),
            tobor.getSonarPacCount(),
            tobor.getLeftVel(),
            tobor.getRightVel(),
            byte_as_bitstring(tobor.getDigIn()).c_str(),
            byte_as_bitstring(tobor.getDigOut()).c_str(),
            tobor.getLeftEncoder(),
            tobor.getRightEncoder(),
            wasLeftIRTriggered?"YES": "   ",
            wasRightIRTriggered?"YES":"   "
    );


if (!(tobor.getDigIn()&32)){
// prova a fare beep
            tobor.setVel(-200);
            tobor.unlock();
            tobor.lock();
            char tune[]={5,2,5,3,5,4,5,5,5,6,5,7,15,8};
            tobor.comStrN(ArCommands::SAY,tune,14);
            ArUtil::sleep(2000);
```

```
                tobor.unlock();
                tobor.lock();
                tobor.setVel(100);
            }

            if ((tobor.getDigIn()&128)&&pinzagiu){
            // prova a fare beep
                char tune[6]={5,20,25,62,25,64};
                tobor.comStrN(ArCommands::SAY,tune,2);
            }

            if ((tobor.getDigIn()&64)&&pinzagiu){
                pinza.gripOpen(); //che invece la chiude
            // prova a fare beep
                char tune[6]={100,30,25,62,25,64};
                tobor.comStrN(ArCommands::SAY,tune,2);
                pinzagiu=false;
            }


// list indices of bumpers flaged in stallval
// skip the last bit which is a motor stall flag
            ArTypes::UByte2 bumpmask = ArUtil::BIT15;
            int bump = 0;
            for(int bit = 16; bit > 0; bit--)
```

```
		{
			if(bit == 9) // this is also a motor stall bit
			{
				bumpmask = bumpmask >> 1;
				bit--;
				continue;
			}
//printf("\n\tComparing stallval=%s to bumpmask=%s... ", int_as_bitstring(stallval).c_str(),
int_as_bitstring(bumpmask).c_str());
			if(cumulativeStallVal & bumpmask)
				printf("%d ", bump);
			bumpmask = bumpmask >> 1;
			bump++;
		}

// print pose of last bump sensor reading
		const std::list<ArPoseWithTime*>* bumpsensed = bumpers.getCurrentBuffer();
		if(bumpsensed)
		{
//printf("%d readings. ", bumpsensed->size());
			if(bumpsensed->size() > 0 && bumpsensed->front()) {
				printf("(%.0f,%.0f)", bumpsensed->front()->getX(), bumpsensed->front()->getY());
			}
		}
		puts("");
```

```
// clear events to accumulate for the next second
                cumulativeRobotFlags = cumulativeStallVal = 0;
                wasLeftMotorStalled = wasRightMotorStalled = wasLeftIRTriggered = wasRightIRTriggered =
wasEStopTriggered = false;

                tobor.unlock();
                ArUtil::sleep(1000);
        }

// robot cycle stopped, probably because of lost robot connection
        tobor.lock();
        pinza.liftDown(); //Apre la pinza
        tobor.unlock();
Aria::shutdown();
        return 0;
}
```

Appendice 2

Il programma “Caramelle.cpp” dopo le modifiche

```
#include "caramelle.h"

#include <string>

/* function to display a byte as a string of 8 '1' and '0' characters. */

std::string byte_as_bitstring(char byte)

{

	char tmp[9];

	int bit;

	int ch;

	for(bit = 7, ch = 0; bit >= 0; bit--,ch++)

		tmp[ch] = ((byte>>bit)&1) ? '1' : '0';

	tmp[8] = 0;

	return std::string(tmp);

}



/* function to display a 2-byte int as a string of 16 '1' and '0' characters. */

std::string int_as_bitstring(ArTypes::Byte2 n)

{

	char tmp[17];

	int bit;

	int ch;

	for(bit = 15, ch = 0; bit >= 0; bit--, ch++)

		tmp[ch] = ((n>>bit)&1) ? '1' : '0';
```

```
	tmp[16] = 0;

	return std::string(tmp);

}


/* Some events might only be detectable in one robot cycle, not over the

* 1-second period that the main thread sleeps. This cycle callback will detect

* those and save them in some global variables. */

bool wasLeftMotorStalled = false;

bool wasRightMotorStalled = false;

ArTypes::UByte2 cumulativeStallVal = 0;

ArTypes::UByte2 cumulativeRobotFlags = 0;

bool wasLeftIRTriggered = false;

bool wasRightIRTriggered = false;

bool wasEStopTriggered = false;


bool cycleCallback(ArRobot* tobor)

{

	cumulativeStallVal |= tobor->getStallValue();

	wasLeftMotorStalled = wasLeftMotorStalled || tobor->isLeftMotorStalled();

	wasRightMotorStalled = wasRightMotorStalled || tobor->isRightMotorStalled();

	wasEStopTriggered = wasEStopTriggered || tobor->getEstop();

	wasLeftIRTriggered = wasLeftIRTriggered || (tobor->hasTableSensingIR() && tobor-
>isLeftTableSensingIRTriggered());

	wasRightIRTriggered = wasRightIRTriggered || (tobor->hasTableSensingIR() && tobor-
>isRightTableSensingIRTriggered());

	return true;
```

```
}


/* main function */
int main(int argc, char **argv)
{
      ArArgumentParser parser(&argc, argv); // new
            parser.loadDefaultArguments();  //  new
// robot and devices
            ArRobot tobor;
            ArSonarDevice sonar;
            ArBumpers bumpers;
            ArIRs ir;

// the actions we'll use to wander and avoid obstacles
            ArActionStallRecover recoverAct;
            //ArActionAvoidSide avoidSide;
            ArActionBumpers bumpAct;
      ArActionAvoidFront avoidFront;
            //ArActionAvoidFront avoidFrontNearAct("Avoid Front Near", 150, 0, 20);
            //ArActionAvoidFront avoidFrontFarAct("Avoid Front Far", 300, 100, 15);
            ArActionConstantVelocity constantVelocityAct("Constant Velocity", 1000);

// initialize aria and aria's logging destination and level
            Aria::init();
```

```
        ArLog::init(ArLog::StdErr, ArLog::Normal);


// connector nuovo al posto del simple
      ArRobotConnector robotConnector(&parser, &tobor);



        printf("This program will make the robot wander around, avoiding obstacles, and print some data and events.\nPress Ctrl-C to exit.\n");

ArLaserConnector laserConnector(&parser, &tobor, &robotConnector); // new

// add the range devices to the robot
        tobor.addRangeDevice(&sonar);
        tobor.addRangeDevice(&bumpers);
        tobor.addRangeDevice(&ir);

        ArGripper pinza=ArGripper(&tobor);

        //metodo nuovo di tentativo di connessione
if (!robotConnector.connectRobot())
{
  // Error connecting:
  // if the user gave the -help argumentp, then just print out what happened,
  // and continue so options can be displayed later.
  if (!parser.checkHelpAndWarnUnparsed())
```

```
  {
    ArLog::log(ArLog::Terse, "Could not connect to robot, will not have parameter file so options displayed later may not include everything");
  }
  // otherwise abort
  else
  {
    ArLog::log(ArLog::Terse, "Error, could not connect to robot.");
    Aria::logOptions();
    Aria::exit(1);
  }
}

 if (!laserConnector.connectLasers(false, false, true))
{
  printf("Could not connect to lasers... exiting\n");
  Aria::exit(2);
}

// turn on the motors, turn off amigobot sound effects (for old h8-model amigobots)
        tobor.enableMotors();
//      tobor.comInt(ArCommands::SOUNDTOG, 0);

// add the actions created above
        tobor.addAction(&recoverAct, 90);
```

```
        tobor.addAction(&bumpAct, 100);
        tobor.addAction(&avoidFront,79);
        //tobor.addAction(&avoidFrontNearAct, 50);
        //tobor.addAction(&avoidFrontFarAct, 31);
        //tobor.addAction(&avoidSide,30);
        tobor.addAction(&constantVelocityAct, 50);


// Cycle callback to check for events
        tobor.addUserTask("checkevents", 1, new ArGlobalRetFunctor1<bool, ArRobot*>(&cycleCallback, &tobor));
ArModeLaser laser(&tobor, "laser", 'l', 'L');


// start the robot running, true means that if we lose robot connection the
// ArRobot runloop stops
        tobor.runAsync(true);
        tobor.lock();
        printf ("Il tipo della pinza e' %d\n\n",pinza.getType() );
        tobor.unlock();


        tobor.lock();
        pinza.liftDown(); //Apre la pinza
        tobor.unlock();


        char pinzagiu=true;
//ArUtil::sleep(10000);
```

```
// Print data header

#define HEADFORMAT "%-24s %-5s %-16s %-5s %-6s %-6s %-16s %-8s %-8s %-8s %-8s %-8s %-8s %-10s %-10s %-5s %-5s %s"

#define DATAFORMAT "%-24s %03.02f %-16s %-5s %-6s %-6s %-16s %-8d %-8d %-8g %-8g %-8s %-8s %-10lu %-10lu %-5s %-5s" // doesn't include bumps details on end

        printf("\n" HEADFORMAT "\n\n",

            "Time",

            "Volts",

            "Flags",

            "EStop",

            "StallL",

            "StallR",

            "StallVal",

            "#SIP/sec",

            "#Son/sec",

            "Vel L",

            "Vel R",

            "DigIns",

            "DigOuts",

            "Enc L",

            "Enc R",

            "IR L",
```

```
                "IR R",
                "Cur Bumps, (Last Bump Pose)"
               );

// Request that we will want encoder data

        tobor.requestEncoderPackets();



// Print data every second
        char timestamp[24];
    int cont = 0;
        while(tobor.isRunning()) {

                tobor.lock();
                time_t t = time(NULL);
                strftime(timestamp, 24, "%Y-%m-%d %H:%M:%S", localtime(&t));
        /*if(cont == 10){
          pinza.gripOpen();
          char tune[]={5,2,5,3,5,4,5,5,5,6,5,7,15,8};
                        tobor.comStrN(ArCommands::SAY,tune,14);
        }
        if(cont == 20){
          pinza.liftDown();
          char tune[6]={5,20,25,62,25,64};
```

```
                    tobor.comStrN(ArCommands::SAY,tune,2);
 cont = 0;
  }
cont++;*/
          printf( DATAFORMAT,
                    timestamp,
                    tobor.getRealBatteryVoltage(),
                    int_as_bitstring(cumulativeRobotFlags).c_str(),
                    (wasEStopTriggered ? "YES" : "   "),
                    (wasLeftMotorStalled?"YES":"   "),
                    (wasRightMotorStalled?"YES":"   "),
                    int_as_bitstring(cumulativeStallVal).c_str(),
                    tobor.getMotorPacCount(),
                    tobor.getSonarPacCount(),
                    tobor.getLeftVel(),
                    tobor.getRightVel(),
                    byte_as_bitstring(tobor.getDigIn()).c_str(),
                    byte_as_bitstring(tobor.getDigOut()).c_str(),
                    tobor.getLeftEncoder(),
                    tobor.getRightEncoder(),
                    wasLeftIRTriggered?"YES": "   ",
                    wasRightIRTriggered?"YES":"   "
                );
       if(cont==10)
            laser.activate();
```

```
                if(cont==20){
                 laser.deactivate();
                 cont=0;
                }
                cont++;
```

```
        if (!(tobor.getDigIn()&32)){
        // prova a fare beep
                tobor.setVel(-200);
                tobor.unlock();
                tobor.lock();
                char tune[]={5,2,5,3,5,4,5,5,5,6,5,7,15,8};
                tobor.comStrN(ArCommands::SAY,tune,14);
                ArUtil::sleep(2000);
                tobor.unlock();
                tobor.lock();
                tobor.setVel(100);
        }

        if ((tobor.getDigIn()&128)&&pinzagiu){
        // prova a fare beep
                char tune[6]={5,20,25,62,25,64};
```

```
                tobor.comStrN(ArCommands::SAY,tune,2);
        }

        if ((tobor.getDigIn()&64)&&pinzagiu){
                pinza.gripOpen(); //che invece la chiude
        // prova a fare beep
                char tune[6]={100,30,25,62,25,64};
                tobor.comStrN(ArCommands::SAY,tune,2);
                pinzagiu=false;
        }

// list indices of bumpers flaged in stallval
// skip the last bit which is a motor stall flag
        ArTypes::UByte2 bumpmask = ArUtil::BIT15;
        int bump = 0;
        for(int bit = 16; bit > 0; bit--)
        {
                if(bit == 9) // this is also a motor stall bit
                {
                        bumpmask = bumpmask >> 1;
                        bit--;
                        continue;
                }
```

```
//printf("\n\tComparing stallval=%s to bumpmask=%s... ", int_as_bitstring(stallval).c_str(),
int_as_bitstring(bumpmask).c_str());

                        if(cumulativeStallVal & bumpmask)

                                printf("%d ", bump);

                        bumpmask = bumpmask >> 1;

                        bump++;

                }


// print pose of last bump sensor reading

                const std::list<ArPoseWithTime*>* bumpsensed = bumpers.getCurrentBuffer();

                if(bumpsensed)

                {

//printf("%d readings. ", bumpsensed->size());

                        if(bumpsensed->size() > 0 && bumpsensed->front()) {

                                printf("(%.0f,%.0f)", bumpsensed->front()->getX(), bumpsensed->front()->getY());

                        }

                }

                puts("");


// clear events to accumulate for the next second

                cumulativeRobotFlags = cumulativeStallVal = 0;

                wasLeftMotorStalled = wasRightMotorStalled = wasLeftIRTriggered = wasRightIRTriggered =
wasEStopTriggered = false;


                tobor.unlock();

                ArUtil::sleep(1000);
```

```
        }


// robot cycle stopped, probably because of lost robot connection


        tobor.lock();

        pinza.liftDown(); //Apre la pinza

        tobor.unlock();


        Aria::shutdown();

        return 0;

}
```

Appendice 3

Versione iniziale del file “pion1m.p”

```
;SectionFlags for :

;  Robot parameter file


Section General settings

;SectionFlags for General settings:

Class Pioneer          ; general type of robot

Subclass pion1m        ; specific type of robot

RobotRadius 220        ; radius in mm

RobotDiagonal 90       ; half-height to diagonal of octagon

RobotWidth 400         ; width in mm

RobotLength 500        ; length in mm of the whole robot

RobotLengthFront 0     ; length in mm to the front of the robot (if this is 0

                       ; (or non existant) this value will be set to half of

                       ; RobotLength)

RobotLengthRear 0      ; length in mm to the rear of the robot (if this is 0

                       ; (or non existant) this value will be set to half of

                       ; RobotLength)

Holonomic true         ; turns in own radius

MaxRVelocity 100       ; absolute maximum degrees / sec

MaxVelocity 400        ; absolute maximum mm / sec

MaxLatVelocity 0       ; absolute lateral maximum mm / sec

HasMoveCommand false   ; has built in move command
```

RequestIOPackets false ; automatically request IO packets

RequestEncoderPackets false ; automatically request encoder packets

SwitchToBaudRate 0 ; switch to this baud if non-0 and supported on robot

Section Conversion factors

;SectionFlags for Conversion factors:

AngleConvFactor 0.0061359 ; radians per angular unit (2PI/4096)

DistConvFactor 0.05066 ; multiplier to mm from robot units

VelConvFactor 2.5332 ; multiplier to mm/sec from robot units

RangeConvFactor 0.1734 ; multiplier to mm from sonar units

DiffConvFactor 0.00333333 ; ratio of angular velocity to wheel velocity (unused

; in newer firmware that calculates and returns this)

Vel2Divisor 4 ; divisor for VEL2 commands

GyroScaler 1.626 ; Scaling factor for gyro readings

Section Accessories the robot has

;SectionFlags for Accessories the robot has:

TableSensingIR false ; if robot has upwards facing table sensing IR

NewTableSensingIR false ; if table sensing IR are sent in IO packet

FrontBumpers false ; if robot has a front bump ring

NumFrontBumpers 0 ; number of front bumpers on the robot

RearBumpers false ; if the robot has a rear bump ring

NumRearBumpers 0 ; number of rear bumpers on the robot

Section Sonar parameters

;SectionFlags for Sonar parameters:

SonarNum 7 ; number of sonar on the robot

; SonarUnit <sonarNumber> <x position, mm> <y position, mm> <heading of disc,

; degrees>

SonarUnit 0 100 100 90

SonarUnit 1 120 80 30

SonarUnit 2 130 40 15

SonarUnit 3 130 0 0

SonarUnit 4 130 -40 -15

SonarUnit 5 120 -80 -30

SonarUnit 6 100 -100 -90

Section IR parameters

;SectionFlags for IR parameters:

IRNum 0 ; number of IRs on the robot

; IRUnit <IR Number> <IR Type> <Persistance, cycles> <x position, mm> <y

; position, mm>

Section Movement control parameters

; if these are 0 the parameters from robot flash will be used, otherwise these

; values will be used

;SectionFlags for Movement control parameters:

SettableVelMaxes true ; if TransVelMax and RotVelMax can be set

TransVelMax 400 ; maximum desired translational velocity for the robot

RotVelMax 100 ; maximum desired rotational velocity for the robot

SettableAccsDecs false ; if the accel and decel parameters can be set

TransAccel 0 ; translational acceleration

TransDecel 0 ; translational deceleration

RotAccel 0 ; rotational acceleration

RotDecel 0 ; rotational deceleration

HasLatVel false ; if the robot has lateral velocity

LatVelMax 0 ; maximum desired lateral velocity for the robot

LatAccel 0 ; lateral acceleration

LatDecel 0 ; lateral deceleration

Appendice 4

Versione modificata del file “pion1m.p”

```
;SectionFlags for :
;  Robot parameter file

Section General settings
;SectionFlags for General settings:
Class Pioneer          ; general type of robot
Subclass pion1m        ; specific type of robot
RobotRadius 220        ; radius in mm
RobotDiagonal 90       ; half-height to diagonal of octagon
RobotWidth 400         ; width in mm
RobotLength 500        ; length in mm of the whole robot
RobotLengthFront 0     ; length in mm to the front of the robot (if this is 0
                       ; (or non existant) this value will be set to half of
                       ; RobotLength)
RobotLengthRear 0      ; length in mm to the rear of the robot (if this is 0
                       ; (or non existant) this value will be set to half of
                       ; RobotLength)
Holonomic true         ; turns in own radius
MaxRVelocity 100       ; absolute maximum degrees / sec
MaxVelocity 400        ; absolute maximum mm / sec
MaxLatVelocity 0       ; absolute lateral maximum mm / sec
```

HasMoveCommand false ; has built in move command

RequestIOPackets false ; automatically request IO packets

RequestEncoderPackets false ; automatically request encoder packets

SwitchToBaudRate 0 ; switch to this baud if non-0 and supported on robot

Section Conversion factors

;SectionFlags for Conversion factors:

AngleConvFactor 0.0061359 ; radians per angular unit (2PI/4096)

DistConvFactor 0.05066 ; multiplier to mm from robot units

VelConvFactor 2.5332 ; multiplier to mm/sec from robot units

RangeConvFactor 0.1734 ; multiplier to mm from sonar units

DiffConvFactor 0.00333333 ; ratio of angular velocity to wheel velocity (unused

; in newer firmware that calculates and returns this)

Vel2Divisor 4 ; divisor for VEL2 commands

GyroScaler 1.626 ; Scaling factor for gyro readings

Section Accessories the robot has

;SectionFlags for Accessories the robot has:

TableSensingIR false ; if robot has upwards facing table sensing IR

NewTableSensingIR false ; if table sensing IR are sent in IO packet

FrontBumpers false ; if robot has a front bump ring

NumFrontBumpers 0 ; number of front bumpers on the robot

RearBumpers false ; if the robot has a rear bump ring

NumRearBumpers 0 ; number of rear bumpers on the robot

Section Sonar parameters

;SectionFlags for Sonar parameters:

;SonarNum 7 ; number of sonar on the robot

SonarNum8

; SonarUnit <sonarNumber> <x position, mm> <y position, mm> <heading of disc,

; degrees>

SonarUnit 0 100 100 90

SonarUnit 1 120 80 30

SonarUnit 2 130 40 15

SonarUnit 3 130 0 0

SonarUnit 4 130 -40 -15

SonarUnit 5 120 -80 -30

SonarUnit 6 100 -100 -90

;SonarNum 7 ; number of sonar on the robot

SonarUnit 7 0 0 180

Section IR parameters

;SectionFlags for IR parameters:

IRNum 0 ; number of IRs on the robot

; IRUnit <IR Number> <IR Type> <Persistance, cycles> <x position, mm> <y

; position, mm>

Section Movement control parameters

; if these are 0 the parameters from robot flash will be used, otherwise these

; values will be used

;SectionFlags for Movement control parameters:

SettableVelMaxes true ; if TransVelMax and RotVelMax can be set

TransVelMax 1000 ; maximum desired translational velocity for the robot

RotVelMax 1000 ; maximum desired rotational velocity for the robot

;SettableAccsDecs false ; if the accel and decel parameters can be set

SettableAccsDecs true

TransAccel 1000 ; translational acceleration

TransDecel 1000 ; translational deceleration

RotAccel 1000 ; rotational acceleration

RotDecel 1000 ; rotational deceleration

HasLatVel false ; if the robot has lateral velocity

LatVelMax 0 ; maximum desired lateral velocity for the robot

LatAccel 0 ; lateral acceleration

LatDecel 0 ; lateral deceleration

Appendice 5

La CGI “tourGoals.cgi”

```
#!/bin/bash

echo "Status: 204 No Change”

echo

echo

elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Tobor_on2

killall -9 elinks

/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null

/usr/local/Arnl/examples/arnlServer3 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial

-connectLaser >/dev/null &

sleep 15

/usr/local/Arnl/examples/goalClient1 >/dev/null

/usr/local/Arnl/examples/goalClient1 >/dev/null

/usr/local/Arnl/examples/goalClient2 >/dev/null

/usr/local/Arnl/examples/goalClient2 >/dev/null

/usr/local/Arnl/examples/goalClient3 >/dev/null

/usr/local/Arnl/examples/goalClient3 >/dev/null

/usr/local/Arnl/examples/goalClient4 >/dev/null

/usr/local/Arnl/examples/goalClient4 >/dev/null

/usr/local/Arnl/examples/goalClient5 >/dev/null

/usr/local/Arnl/examples/goalClient5 >/dev/null

/usr/local/Arnl/examples/goalClient6 >/dev/null

/usr/local/Arnl/examples/goalClient6 >/dev/null

/usr/local/Arnl/examples/goalClient7 >/dev/null

/usr/local/Arnl/examples/goalClient7 >/dev/null

/usr/local/Arnl/examples/goalClient0 >/dev/null
```

```
/usr/local/Arnl/examples/goalClient0 >/dev/null

/usr/local/Arnl/examples/goalClient0 >/dev/null

elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative

killall -9 elinks

killall -9 arnlServer3

/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null

acpi -a >/home/user/programmi_robot/state2.txt

if grep -q off-line /home/user/programmi_robot/state2.txt

then

/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null

/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial

-connectLaser >/dev/null &

sleep 20

/usr/local/Arnl/examples/goalClient0 >/dev/null

elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative

killall -9 arnlServer2

/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null

fi

acpi -a >/home/user/programmi_robot/state2.txt

if grep -q off-line /home/user/programmi_robot/state2.txt

then

/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null

/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial

-connectLaser >/dev/null &

sleep 20

/usr/local/Arnl/examples/goalClient0 >/dev/null

elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative

killall -9 arnlServer2

/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
```

```
fi
acpi -a >/home/user/programmi_robot/state2.txt
if grep -q off-line /home/user/programmi_robot/state2.txt
then
/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null
/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial
-connectLaser >/dev/null &
sleep 20
/usr/local/Arnl/examples/goalClient0 >/dev/null
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative
killall -9 arnlServer2
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
acpi -a >/home/user/programmi_robot/state2.txt
if grep -q off-line /home/user/programmi_robot/state2.txt
then
/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null
/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial
-connectLaser >/dev/null &
sleep 20
/usr/local/Arnl/examples/goalClient0 >/dev/null
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative
killall -9 arnlServer2
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
acpi -a >/home/user/programmi_robot/state2.txt
if grep -q off-line /home/user/programmi_robot/state2.txt
then
/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null
```

```
/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial
-connectLaser >/dev/null &
sleep 20
/usr/local/Arnl/examples/goalClient0 >/dev/null
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative
killall -9 arnlServer2
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?End_movement
killall -9 elinks
```

NOTA. La doppia invocazione del programma “elinks”, seguita da un indirizzo web, provoca l’avvio di un messaggio sonoro di avviso all’interno del laboratorio sia all’inizio, sia alla fine dei movimenti. Inoltre si è resa necessaria una doppia invocazione di programma client per ogni goal (punto della mappa da visitare). Infatti, nel caso un goal non fosse raggiungibile, per come sono stati scritti i programmi client, non verrebbe saltato solo quello, ma anche il successivo. Si è reso inoltre indispensabile indirizzare l’output prodotto da ogni programma invocato (delle stampe a video sono sempre presenti) su un device nullo. Questo garantisce il corretto funzionamento della CGI.

Appendice 6

Il programma “goalClient1.cpp”

```
#include "Aria.h"

#include "ArNetworking.h"



void handlePathPlannerStatus(ArNetPacket *packet)

{

  char buf[64];

  packet->bufToStr(buf, 63);

  printf(".. Path planner status: \"%s\"\n", buf);

  if(strcmp(buf,"Failed going to goal")==0 || strcmp(buf, "Failed to plan to 1")==0 || strcmp(buf,"Cannot find
path")==0)

  exit(0);

}



void handleGoalName(ArNetPacket* packet)

{

  char buf[64];

  packet->bufToStr(buf, 63);

  printf(".. Current goal: \"%s\"\n", buf);

}
```

```
void handleRobotUpdate(ArNetPacket* packet)
{
  char buf[64];
  packet->bufToStr(buf, 63);
  printf(".. Robot server status: \"%s\"\n", buf);
  if(strcmp("Arrived at 1", buf)==0 || strcmp("Failed to get to 1 (Failed going to goal)",buf)==0 || strcmp(buf, "Failed to plan to 1")==0 || strcmp(buf, "F$
    exit(0);
  packet->bufToStr(buf, 63);
  printf(".. Robot server mode: \"%s\"\n", buf);
}



void handleGoalList(ArNetPacket *packet)
{
  printf(".. Server has these goals:\n");
  char goal[256];
  for(int i = 0; packet->getReadLength() < packet->getLength(); i++)
  {
    packet->bufToStr(goal, 255);
    if(strlen(goal) == 0)
        return;
    printf("      %s\n", goal);
  }
```

```
}

int main(int argc, char **argv)
{
  Aria::init();
  ArClientBase client;
  ArArgumentParser parser(&argc, argv);
  ArClientSimpleConnector clientConnector(&parser);
  parser.loadDefaultArguments();

  if (!clientConnector.parseArgs() ||
!parser.checkHelpAndWarnUnparsed())
  {
    clientConnector.logOptions();
    exit(0);
  }

  printf("Connecting...\n");
  if (!clientConnector.connectClient(&client))
  {
    if (client.wasRejected())
      printf("Server rejected connection, exiting\n");
    else
      printf("Could not connect to server, exiting\n");
```

```
    exit(1);
  }

  printf("Connected to server.\n");
  client.addHandler("pathPlannerStatus", new
ArGlobalFunctor1<ArNetPacket*>(&handlePathPlannerStatus));
  client.addHandler("update", new
ArGlobalFunctor1<ArNetPacket*>(&handleRobotUpdate));
  client.addHandler("goalName", new
ArGlobalFunctor1<ArNetPacket*>(&handleGoalName));
  client.addHandler("getGoals", new
ArGlobalFunctor1<ArNetPacket*>(&handleGoalList));
  client.runAsync();
  client.requestOnce("getGoals");
  client.request("pathPlannerStatus", 5000);
  client.request("goalName", 5000);
  client.request("update", 5000);
 printf("=> Sending goal \"%s\" to server...\n", "1");
      client.requestOnceWithString("gotoGoal", "1");
      sleep(1000);

  printf("Server disconnected.\n");
  Aria::shutdown();
  return 0;
}
```

Appendice 7

La CGI “wander.cgi”

```
#!/bin/bash


echo "Status: 204 No Change"

echo

echo

/usr/local/Arnl/examples/wander >/dev/null

elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Tobor_on2

killall -9 elinks

/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null

/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial

-connectLaser >/dev/null &

sleep 200

/usr/local/Arnl/examples/goalClient0 >/dev/null

elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative

killall -9 arnlServer2

/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null

acpi -a >/home/user/programmi_robot/state2.txt

if grep -q off-line /home/user/programmi_robot/state2.txt

then

/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null

/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial

-connectLaser >/dev/null &

sleep 20

/usr/local/Arnl/examples/goalClient0 >/dev/null

elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative

killall -9 arnlServer2
```

```
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
acpi -a >/home/user/programmi_robot/state2.txt
if grep -q off-line /home/user/programmi_robot/state2.txt
then
/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null
/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial
-connectLaser >/dev/null &
sleep 20
/usr/local/Arnl/examples/goalClient0 >/dev/null
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative
killall -9 arnlServer2
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
acpi -a >/home/user/programmi_robot/state2.txt
if grep -q off-line /home/user/programmi_robot/state2.txt
then
/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null
/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial
-connectLaser >/dev/null &
sleep 20
/usr/local/Arnl/examples/goalClient0 >/dev/null
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative
killall -9 arnlServer2
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
acpi -a >/home/user/programmi_robot/state2.txt
if grep -q off-line /home/user/programmi_robot/state2.txt
then
```

```
/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null
/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial
-connectLaser >/dev/null &
sleep 20
/usr/local/Arnl/examples/goalClient0 >/dev/null
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative
killall -9 arnlServer2
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
acpi -a >/home/user/programmi_robot/state2.txt
if grep -q off-line /home/user/programmi_robot/state2.txt
then
/home/user/programmi_robot/uscitaTobor/uscitaTobor -rp /dev/ttyUSB0 >/dev/null
/usr/local/Arnl/examples/arnlServer2 -rp /dev/ttyUSB0 -map chopper.map -laserType urg -lp /dev/ttyACM0 -lpt serial
-connectLaser >/dev/null &
sleep 20
/usr/local/Arnl/examples/goalClient0 >/dev/null
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?Morgul_on.wav.alternative
killall -9 arnlServer2
/home/user/programmi_robot/rientroTobor/rientroTobor -rp /dev/ttyUSB0 >/dev/null
fi
elinks http://frost.ing.unibs.it/cgi-bin/Activate_sounds.cgi?End_movement
killall -9 elinks
```

Appendice 8

La CGI “stop.cgi”

```
#!/bin/bash


echo "Status: 204 No Change"

echo

echo

/usr/local/Arnl/examples/stop >/dev/null

killall -9 wander.cgi

killall -9 tourGoals.cgi

killall -9 drive.cgi

killall -9 parkRobot.cgi

killall -9 wander

killall -9 goalClient1

killall -9 goalClient2

killall -9 goalClient3

killall -9 goalClient4

killall -9 goalClient5

killall -9 goalClient6

killall -9 goalClient7

killall -9 goalCliantHome

killall -9 rientroTobor

killall -9 destra.cgi

killall -9 sinistra.cgi

killall -9 avanti.cgi

killall -9 indietro.cgi

/usr/local/Arnl/examples/stop >/dev/null
```

Appendice 9

La CGI “state.cgi”

```
#!/bin/bash


echo -e "Content-type: text/html\n"

echo -e "\n\n"

echo -e "<html>\n"

echo -e "<head>\n"

echo -e "<meta http-equiv=\"refresh\" content=\"5\">"

echo -e "</head>\n"

echo -e "<body bgcolor=\"#33CCFF\" text=\"yellow\">\n"

echo -e "<span class=\"titlebar\"><DIV align=\"center\"><FONT color=\"black\" size=\"+1\">This pane reflects Tobor

status, updated every 5 seconds.</FONT></$echo -e "<br>"

echo -e "<br>"

echo -e "<br>"

#echo -e "<P ALIGN=\"CENTER\"> <img src=\"http://192.0.2.24:8082/Mappa.png\" width=\"250\" height=\"180\">

</P>"

echo -e "<table border=1 width=\"400\" bgcolor=\"#0000CC\">"

echo -e "<tr>"

echo -e "<td>"

echo -e "<Font size=\"+1\">Tobor says: </FONT>"

echo -e "<table border=1 width='100%'><TR><TD><Font size=\"+1\"> Name: Tobor </FONT></TD></TR>"

echo -e "<TR>"

/usr/local/Arnl/examples/goalClientStatus1 >/home/user/programmi_robot/state.txt

if grep -q off... /home/user/programmi_robot/state.txt

then echo -e "<TD bgcolor=red><Font size=\"+1\">Robot turned off... </FONT>"

else
```

```
   echo -e "<TD bgcolor=green><Font size=\"+1\">"
fi
grep status /home/user/programmi_robot/state.txt
echo -e "</FONT></TD>"
echo -e "</TR>"
echo -e "<TR>"
/usr/local/Arnl/examples/goalClientStatus2 >/home/user/programmi_robot/state.txt
if grep -q off... /home/user/programmi_robot/state.txt
then echo -e "<TD bgcolor=red><Font size=\"+1\">Robot turned off...</FONT>"
else
   echo -e "<TD bgcolor=green><Font size=\"+1\">"
fi
grep goal /home/user/programmi_robot/state.txt
echo -e "</FONT></TD>"
echo -e "</TR>"
echo -e "<TR>"
/usr/local/Arnl/examples/goalClientStatus3 >/home/user/programmi_robot/state.txt
if grep -q off... /home/user/programmi_robot/state.txt
then echo -e "<TD bgcolor=red><Font size=\"+1\">Robot turned off...</FONT>"
else
   echo -e "<TD bgcolor=green><Font size=\"+1\">"
fi
grep status /home/user/programmi_robot/state.txt
echo -e "</FONT></TD>"
echo -e "</TR>"
echo -e "<TR>"
/usr/local/Arnl/examples/goalClientStatus4 >/home/user/programmi_robot/state.txt
if grep -q off... /home/user/programmi_robot/state.txt
then echo -e "<TD bgcolor=red><Font size=\"+1\">Robot turned off...</FONT>"
```

```
else
   echo -e "<TD bgcolor=green><Font size=\"+1\">"
fi
grep mode /home/user/programmi_robot/state.txt
echo -e "</FONT></TD>"
echo -e "</TR>"
echo -e "<TR>"
echo -e "<TD><Font size=\"+1\">"
acpi
echo -e "</FONT></TD>"
echo -e "</TR>"
echo -e "<TR>"
acpi -a >/home/user/programmi_robot/state.txt
if grep -q off-line /home/user/programmi_robot/state.txt
then echo -e "<TD bgcolor=red><Font size=\"+1\">Robot not docked...</FONT>"
else
   echo -e "<TD bgcolor=green><Font size=\"+1\">Robot docked..."
fi
echo -e "</FONT></TD>"
echo -e "</TR>"
echo -e "</td>"
echo -e "</tr>"
echo -e "</table>"
echo -e "</body>\n"
echo -e "</html>\n"
```

Bibliografia/sitografia

- http://www.hokuyo-aut.jp/
- Documentazione interna alla libreria Aria
- Documentazione interna alla libreria Arnl